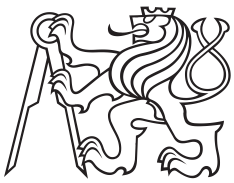
Bachelor's thesis



Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Bicycle Simulator

Richard Dobřichovský
Open Informatics – Computer and Information Science

May 2013
Supervisor: Ing. Petr Felkel, Ph.D.

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

BACHELOR PROJECT ASSIGNMENT

Student: Richard D o b ř i c h o v s k ý

Study programme: Open Informatics

Specialisation: Computer and Information Science

Title of Bachelor Project: Bicycle Simulator

Guidelines:

1. Create a 3D model of a bicycle and its rider suitable for real time rendering. The model will include rider animations and will be made using Autodesk Maya.
2. Implement a FBX model loader using the FBX SDK from Autodesk, Inc. supporting the features required for the project.
3. Study the physics of bicycle moition and create an engine simulating a bicycle ride, including collision detection in complex terrain and rider fatigue. The bicycle will be controlled using a joystick/steering wheel.
4. Create three different maps to demonstrate the simulator engine - a velodrome, a road with numerous bends and offroad terrain.
5. Combine the above-mentioned tasks in an application written in C++ using OpenGL for rendering.
6. Compare the application output with a video of a real rider in typical situations and discuss the results.

Bibliography/Sources:

[1] Autodesk Inc. – Autodesk Maya 8.5 Character Setup – 2007
[2] Autodesk Inc. – Autodesk FBX 2013 SDK Documentation – 2012, dostupné online http://docs.autodesk.com/FBX/2013/ENU/FBX-SDK-Documentation/index.html
[3] Moore, J. K., Human Control of a Bicycle, UC Davis Doctoral Dissertation, 2012, dostupné online http://moorepants.github.com/dissertation/

Bachelor Project Supervisor: Ing. Petr Felkel, Ph.D.

Valid until: the end of the winter semester of academic year 2013/2014

L.S.

prof. Ing. Vladimír Mařík, DrSc.
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, January 10, 2013

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: Richard Dobřichovský

Studijní program: Otevřená informatika (bakalářský)

Obor: Informatika a počítačové vědy

Název tématu: Simulátor jízdy na kole

Pokyny pro vypracování:

1. Vytvořte 3D model jízdního kola a jeho jezdce vhodný pro vykreslování v reálném čase v programu Autodesk Maya, včetně animací jezdce.
2. Implementujte loader modelů formátu FBX pomocí FBX SDK od společnosti Autodesk, Inc. podporující komponenty potřebné pro projekt.
3. Nastudujte fyziku jízdy na kole a vytvořte engine, který ji bude simulovat, včetně detekcí kolizí v členitém terénu a únavy jezdce. Kolo bude ovládané pomocí joysticku či volantu.
4. Vytvořte tři různé mapy k předvedení simulátoru: velodrom, silnici se serpentinami a členitější terén mimo silnici.
5. Zkombinujte výše uvedené body v aplikaci psané v C++ využívající k vykreslování OpenGL.
6. Porovnejte výstup aplikace s videem jezdce v typických situacích a zhodnoťte výsledky.

Seznam odborné literatury:

[1] Autodesk Inc. – Autodesk Maya 8.5 Character Setup – 2007
[2] Autodesk Inc. – Autodesk FBX 2013 SDK Documentation – 2012, dostupné online http://docs.autodesk.com/FBX/2013/ENU/FBX-SDK-Documentation/index.html
[3] Moore, J. K., Human Control of a Bicycle, UC Davis Doctoral Dissertation, 2012, dostupné online http://moorepants.github.com/dissertation/

Vedoucí bakalářské práce: Ing. Petr Felkel, Ph.D.

Platnost zadání: do konce zimního semestru 2013/2014

prof. Ing. Vladimír Mařík, DrSc.
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 10. 1. 2013

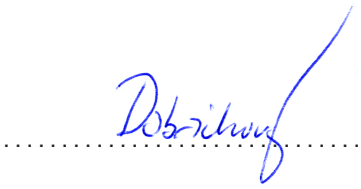
Acknowledgements / Declaration

I would like to thank my advisor, Ing. Petr Felkel, Ph.D. for his constructive approach and advice. I would also like to thank my friend Antonín Kvasnička for his assistance with recording comparison videos. Finally, I would like to thank my parents for their full support during my studies and, in respect of the thesis, my father for taking the photographs for body texturing.

I hereby declare that I have completed this thesis independently and that I have listed all the literature and publications used.

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

In Prague, the 24th of May, 2013



.....

Abstrakt / Abstract

Cílem této práce je vytvoření simulátoru jízdy na kole – počítačové aplikace umožňující uživateli ovládat jezdce na kole na různých tratích. Herní studia se tomuto tématu doposud příliš nevěnovala, a proto jsem se rozhodl jím zabývat. Práce zkoumá fyzikální zákony, které jízdu ovlivňují a nabyté poznatky shrnuje, aby mohly být následně převedeny do fyzikálního enginu a doplněny o detekci kolizí. Práce se zabývá volbou ovladače pro uživatele a dále se soustředí na tvorbu grafických prvků pro simulátor, konkrétně vlastního jízdního kola, jeho jezdce včetně animací, a v poslední řadě tratí. Pro interpretaci těchto modelů je implementována třída pro načítaní 3D modelů ve formátu FBX. Všechny součásti jsou spojeny ve zobrazovacím řetězci pro vykreslování výsledného obrazu pro uživatele a nakonec jsou výstupy porovnány se sadou videí skutečného jezdce na kole. Videa ukazují, že je aplikace schopna věrně simulovat jízdu na kole ve zjednodušeném prostředí virtuální reality.

Klíčová slova:
Jízdní kolo, simulátor, fyzika, Maya, animace, FBX, OpenGL.

Překlad titulu: Simulátor jízdy na kole

The goal of this thesis is to create a bicycle simulator – a computer application enabling its user to control a bicycle rider on different tracks. So far there has been only little interest in the topic by the gaming industry and therefore I take the opportunity to explore a new field. The thesis studies the physics that are involved in bicycle motion and presents the findings to be later converted into a physics engine together with collision detection. It justifies the choice of a controller for human input and further focuses on the creation of graphical elements necessary for the project, namely the bicycle itself, the rider including animations and lastly the ridden tracks. To interpret these, an FBX 3D model loader is implemented and all the components are combined within an OpenGL rendering framework to facilitate screen output to the user. Finally, the outcome is compared with a set of videos of a real cyclist. The videos show that the application is capable of closely simulating a bicycle ride in a simplified virtual reality environment.

Keywords:
Bicycle, simulator, physics, Maya, animation, FBX, OpenGL.

Contents

1 Introduction 1
1.1 Motivation 1
1.2 Existing Implementations 1
1.3 Vision 2
1.4 Outline 2
1.4.1 Bicycle physics 2
1.4.2 User Input 3
1.4.3 Bicycle and Rider 3D Model and Animations 3
1.4.4 3D Model Loading 3
1.4.5 Tracks, Collisions and Physics Engine 3
1.4.6 Rendering Framework 3
2 Bicycle Physics 4
2.1 The Bicycle Model 4
2.2 Bicycle Balance 5
2.3 Riding through Curves 7
2.3.1 Path 7
2.3.2 Lean 8
2.3.3 Countersteering 9
2.4 Sideslip 10
2.4.1 Banked Turn 11
2.4.2 Coefficients of Friction 12
2.5 Resistance 13
2.5.1 Drivetrain and Bearings 13
2.5.2 Rolling Resistance 13
2.5.3 Sloped Terrain 13
2.5.4 Air Drag 13
2.6 Acceleration and Braking 14
2.6.1 Acceleration 14
2.6.2 Braking 15
2.7 Physiological Properties of Rider 15
3 User Input 18
3.1 The Joystick 18
3.2 The Steering Wheel 19
3.3 Critical Situations 19
3.4 Gradual Leaning 19
3.5 Proposed Ideal Controller 20
4 3D Modelling and Animation 21
4.1 The Bicycle 21
4.2 The Rider 22
4.2.1 Body and Texturing 22
4.2.2 Animation 23
4.3 Tracks 24
4.3.1 The Velodrome 24
4.3.2 Road with Bends 26
4.3.3 Offroad Terrain 27
5 3D Model Loading 28
5.1 The FBX Format 28
5.2 Implemented Features 29
6 Physics Engine and Collision Detection 30
6.1 Dynamics 30
6.2 Collisions 30
6.2.1 Height Information Extraction 31
6.2.2 Collision Detection 31
6.2.3 Sideslip Evaluation 32
7 Rendering Framework 33
7.1 Libraries 33
7.2 Vertex Data Loading 33
7.3 Rendering 34
7.4 GLSL Shaders 35
7.4.1 General Rendering 35
7.4.2 Heads-up Display 35
8 Video Comparison 36
9 Discussion 37
10 Conclusion 38
References 39
A Implementation 41
A.1 Files and Classes 41
A.2 Compilation 41
A.3 User Manual 41
B Enclosed CD 43
C List of Symbols and Abbreviations 44
C.1 Symbols 44
C.2 Abbreviations 45

Tables / Figures

2.1. Basic parameters of a bicycle 5
2.2. Axes of rotation on bicycle 5
2.3. Coefficients of friction 13
2.4. Drag coefficients and frontal area 14

2.1. The representation of a bicycle as four rigid bodies 4
2.2. Axes of rotation on bicycle 6
2.3. Path of each wheel in a turn 8
2.4. Forces acting on bicycle in a turn 9
2.5. Paths of the centre of mass and the front wheel when countersteering 10
2.6. Distribution of forces in a turn with tyre friction 10
2.7. Distribution of forces in a banked turn 11
2.8. NASA power-duration curve for "healthy men" 15
2.9. Effect of cadence on pedalling power output 16
4.1. Bicycle model 21
4.2. Textured human body 22
4.3. Animation skeleton 24
4.4. The velodrome 25
4.5. The road with bends 26
8.1. Video comparison snapshot 36

Chapter 1
Introduction

Cycling is a wonderful sport providing endless opportunities and forms. As computer technology advanced, most popular sports received many sorts of adaptations in the gaming world, but the industry has not yet been that enthusiastic about cycling.

1.1 Motivation

I myself, on the other hand, am a passionate cyclist and some time ago it came to my mind whether such a computer game exists. The search resulted with no great avail, although various implementations can be found.

There may be many reasons for this. The first could be attractiveness – most people would find it much more fun racing motorbikes or cars at high speeds – situations which are more or less inaccessible for ordinary people in real life and thus opening doors for computer simulation. Secondly, bicycle riding and racing above all reflects the rider's physical abilities and technique, which is not the predominant case in motorsport. Therefore it is much easier to realistically simulate car driving, when controllers such as steering wheels with pedals are readily available and do not differ greatly from those found in real cars.

1.2 Existing Implementations

The only computer game focusing solely on bicycle riding appears to be Mountain Bike Adrenaline, developed by Valcon Games in 2007. Although this game appeared relatively recently, its graphics feel like it was at least 5 years earlier – not coincidently did it receive reviews even below the 50% mark. The game itself though provides a wide range of inspiration, along with the many flaws it has in its physics engine that can be addressed. Of course I cannot expect this thesis to be graphically superior to Mountain Bike Adrenaline, as good graphics alone can pose much more work than the game engine itself. Nevertheless, they are an inseparable part of the thesis.

A bike can also be ridden in the more recent episodes of the Grand Theft Auto series by Rockstar Games, starting with GTA: San Andreas, but the feature goes in-line with the arcade style of the game and can only be considered as a fun add-on.

Another group of games focuses on acrobatics, jumps and BMX riding – examples being Dave Mirra Freestyle BMX (2000) or Mat Hoffman's Pro BMX (2001) – again very much arcade style with the goal being to press the right sequence of keys to perform a trick in mid air.

An interesting find is quite a large amount of cycling themed 2D games, spanning from the late 80s designed for game consoles of the time, through the 90s for consoles such as the Nintendo Gameboy, to present day in the form of web browser flash games or games designed for current mobile devices. It is worth mentioning Bike or Die!, strongly inspired by Elasto Mania (a 2D simulation of jumps and acrobatics on a motorcycle) and developed for PalmOS and later also published for iOS. The game offers interesting and very well

playable (although absolutely unrealistic) physics, but probably the best in all 2D cycling games I have seen.

Recently I came across a 3D game for iOS: DMBX – Mountain Biking (2011) and its sequel DMBX 2 (2012), both developed by Randerline. As the name suggests, it focuses on mountain biking, but unfortunately I have not had the opportunity to test it, though screenshots suggest its theme is very similar to Mountain Bike Adrenaline and the graphics too are comparable, though it is designed for mobile devices.

Because of much higher popularity of cycling games for mobile devices, it could be interesting to create a port of my simulator for a certain mobile platform in the future.

In professional cycling, interactive training tools have emerged in the last couple of years. Before they only incorporated rollers of variable resistance to simulate different sloping and to adjust training intensity, able to show statistics etc. With the advance of technology, this could be accompanied by a video projection of a real recorded race course with variable playback speed according to the speed of the trainee. A similar device to this was also developed as a master's thesis at the Institute of Intermedia at our faculty. Most recently, virtual reality has also been added as an option for indoor trainers by Tacx[1]), but being aimed at the highest levels of bicycle racing it is out of the price range of an average rider.

Lastly, I came across the FIVIS[2]) bicycle simulator developed by the Institute of Visual Computing at the Bonn-Rhein-Sieg University of Applied Sciences which is supposed to be used for traffic safety education and training. Its rider input comes from a modified real life bicycle, but according to their video[3]), the simulator itself only incorporates physics in a flat ground environment – which can be sufficient for the purpose, when its main goal is to demonstrate traffic rules. Their website states, that work is in progress on FIVIS2, an evolution of the current simulator.

1.3 Vision

The result of this thesis should provide the software for a near-realistic simulation of cycling in a virtual reality environment based on an in-depth analysis of the physics governing bicycle motion. It should asses the limitations of human interface devices and ways to overcome these limitations through a combination of input evaluation and rider and bike animation, as well as provide an interface for a possibly more realistic control device in the future. It should offer a selection of three ridable tracks – a velodrome, a sloped road with numerous bends and off-road terrain.

1.4 Outline

The thesis can be broken down into several distinct parts as outlined below.

1.4.1 Bicycle physics

One of the main points of this thesis is simulating the behaviour of a rider on a bike in the real world. Physics as a science field have provided us with sufficient mechanisms to describe various forms of motion, forces or friction which will find their use in describing what is needed for the simulation, discussed in chapter 2.

[1]) `http://www.tacx.com`
[2]) `http://vc.inf.h-bonn-rhein-sieg.de/?page_id=425`
[3]) `http://www.youtube.com/watch?v=JiVIgMfGhqE`

1.4.2 User Input

In order to control the simulated rider and bicycle, human input is required. Interfaces to human input devices are nowadays relatively standardised, so one can replace a steering wheel by a joystick or any other controller with axes without modifications to the software it controls. In the thesis I will be working with both a steering wheel and a joystick and comparing the results in chapter 3. No barriers exist though to building a more realistic controller resembling a real bike, as it has already been done for various purposes.

1.4.3 Bicycle and Rider 3D Model and Animations

The 3D polygonal models were created using Autodesk Maya to be suitable for real time rendering. Apart from the modelling itself, the process includes texturing, skin binding and skeletal animation with the use of a combination of inverse and forward kinematics. In addition to that, Autodesk Mudbox was used for texture painting when creating the rider. The procedure is illustrated in chapter 4.

1.4.4 3D Model Loading

In order to display pre-modelled objects, such as a bicycle, its rider and the surrounding environment, it is necessary to load these models from files, interpret the loaded information and transform it to a format suitable for rendering through OpenGL. The format chosen for the project is FBX from Autodesk, Inc. who provide an extensive SDK for manipulation with the format and its features. Both the FBX file format and the developed loader are described in chapter 5.

1.4.5 Tracks, Collisions and Physics Engine

Apart from the obvious task of modelling each environment in Maya, maps also pose a challenge in collision detection, as for the physics engine to work properly it is necessary to know the characteristics of the terrain that is being ridden through. This is attained by using heightmaps which are created after the tracks are loaded by rendering the track from a top orthographic view and using the OpenGL Z-buffer to determine the heights. This process as well as the evaluation of collisions and other factors influencing the physics engine is presented in chapter 6.

1.4.6 Rendering Framework

This part can be considered as the foundation of the project which synthesises the elements from all other parts and creates the resulting image on the screen. The programming language used is C++ and OpenGL together with GLSL shaders is used for rendering. The rendering framework is discussed in chapter 7.

Chapter 2
Bicycle Physics

Although the ability to ride a bicycle is a skill each human has to learn, it is an easy process that can be completed even by children younger than 4. With the increase of speed, stabilising the bicycle becomes a much easier task and a trained rider does this nearly unconsciously [1]. Apart from this, the design of bicycles has made them capable of self stabilisation at sufficient speeds i.e. steering themselves to adjust lean with no rider input. The mathematics and physics to describe this behaviour are very complex and are still undergoing research at present day, examples of which can be found in [2] and [1]. This description is not required for the purposes of this simulation and these principles can suffice as empirical observations.

2.1 The Bicycle Model

A very nice model representation is given in [2] and throughout this chapter I will use the same notation.

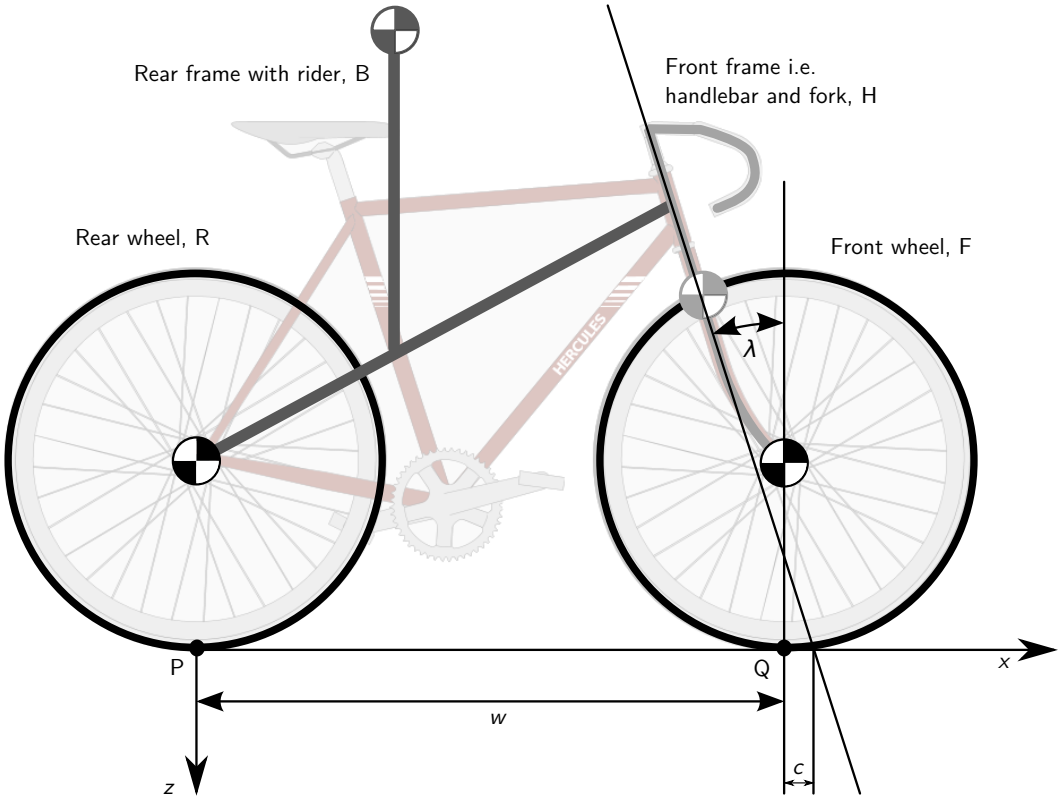


Figure 2.1. The representation of a bicycle as four rigid bodies.

For the purpose of simulation, it is enough to represent a bicycle as four rigid bodies as shown in figure 2.1: the rear wheel R, the rear frame including the rider B, the front frame –

fork and handlebars H and front wheel F. A similar hierarchy will be used in the 3D model, but with adding the details which make a bicycle look like a bicycle. Each of these rigid bodies has its own centre of mass (COM), but for the purpose of our simulation, this will not be necessary, because only the value of mass in total influences the simulation.

Figure 2.1 also shows us some of the basic parameters of a bicycle.

w	wheelbase
c	trail
λ	head tube angle

Table 2.1. Basic parameters of a bicycle.

All three of these parameters determine how much the bicycle itself turns when the rider turns the handlebars, though precisely speaking, the absolute difference is near to minimal. Conventional bikes follow designs that were empirically tested to yield the best results for real-life bicycle handling and have resulted in a range of values commonly used for all conventional bicycle designs. Head tube angles[1]) usually range from 72° on touring bikes to 75° on track bikes, trail commonly ranges from around 35mm on touring bikes to around 75mm on track bikes [3]. When riding a real bicycle, the difference in handling between bikes designed for different purposes (e.g. a track racing bike vs. a touring bike) can be well noticeable as the "feel" or "response" the bicycle has, but it is beyond the possibilities to simulate, because it would require some sort of feedback to the controller, which would have to be very precise.

Crank rotation was not necessary in [2] because they only deal with bicycle stability. For us this value will be important for animation and human power output, as it defines the phase of the rider's legs.

To represent the current state (position and orientation in the environment) of a bicycle, all that is needed are the coordinates of the ground contact point P and the rotation values around 6 axes. Table 2.2 explains these and they are illustrated in figure 2.2.

ψ	direction, or in flight terminology "yaw"
ϕ	lean, or in flight terminology "roll"
θ_B	slope angle, or in flight terminology "pitch"
δ	steer angle
θ_R	rear wheel rotation
θ_F	front wheel rotation
θ_C	crank rotation

Table 2.2. Axes of rotation on bicycle.

2.2 Bicycle Balance

One might think, that a bicycle can ride in a straight line, but unfortunately, that is not the case [4]. Bicycles are single-track vehicles that reside in a labile position when still, therefore prone to leaning to either side. When in motion the lean can be balanced by the rider by steering in its direction [3] which happens all the time, even in near to microscopic changes. The track the bicycle follows in fact represents a sinusoid with its wavelenght differing mainly according to speed [4].

[1]) In bicycle design, the term head tube angle usually refers to the angle of the steering axis and the road. Our λ represents the complementary angle to this.

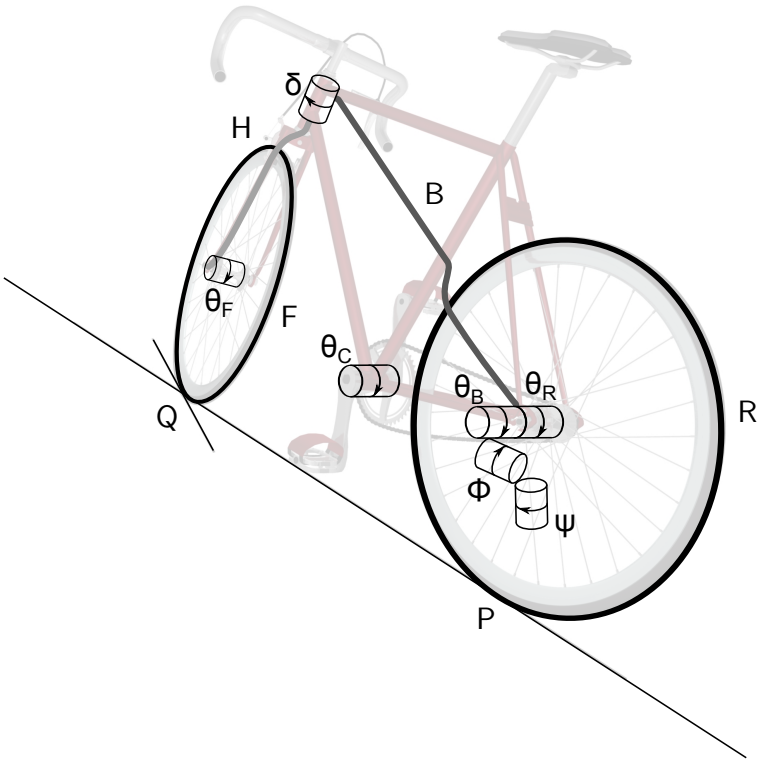


Figure 2.2. Rotation axes on bicycle.

In addition to that, the gyroscopic effect of the front wheel assists the rider in balancing the bicycle by urging the steering in the direction of the lean [3]. This is one of the reasons why balancing a bicycle at higher speeds is much easier. Experiments have been performed with additional gyroscopes both rotating in the same direction as the front wheel to aid balance and rotating against the rotation of the front wheel and thus cancelling its gyroscopic effect [1].

But even a rider at a standstill can balance a bicycle. This is called a "trackstand" and it is a technique sometimes used in track racing when racing head to head, and recently a popular and easy to learn trick for fixed gear riders. In fact the rider is not in a complete standstill: with the handlebars turned to an angle close to 45° the rider balances by moving forward and backwards by very small amounts. In principle, the rider balances the bicycle in exactly the same way as described above. The backwards movement is possible because track bikes have no freewheel but are fixed gear – the rear wheel rotates in respect to the cranks at a fixed gear ratio in both directions. The same is also possible with conventional bikes, if performed on even a slight slope, enabling the gravitational force to initiate the backwards movement.

Although trail plays an important role to make bicycle stabilising easier and is a feature that can be subjectively felt by the rider, it has been proved in [5] that neither positive trail nor the gyroscopic effect of the front wheel is *required* for the bicycle to be able to self-stabilise.

2.3 Riding through Curves

In section 2.1 we discussed various parameters that influence bicycle handling. In this section, the most important will be the variable steer angle δ, the constant wheelbase w and the constant head tube angle λ.

Due to bicycle design, when the handlebars are turned, the angle between the front wheel and the bicycle's forward axis is slightly different. This angle ζ, which is important for determining the radius of a turn, can be easily computed using the parameters mentioned above

$$\zeta = \delta \cos \lambda$$

.

2.3.1 Path

Expressing the turning path is not as easy as it might seem, but it is analogous to that of any ground vehicle able to steer. In short, the front and rear wheel each travel through different paths. The reader can recall tracks left by cars or bicycles on snow in winter, but there is no easy interpretation at first glance.

For explanation, we will consider a steady turn with a fixed steering angle ζ as illustrated in figure 2.3. All unknown values can be computed with the use of goniometric functions. First, it is quite clear, that the turn radius depends on the wheelbase and steering angle and can be computed as

$$r = \frac{w}{\tan \zeta}.$$

According to the laws of circular motion, angular speed is defined as

$$\omega = \frac{v}{r}$$

and therefore the angle covered $\Delta\psi$ can be expressed as

$$\Delta\psi = \frac{v}{r}\Delta t$$

where Δt is the duration of the motion. $\Delta\psi$ expresses the change of the bicycles direction or, in flight terminology, the change in yaw.

With this information, we can calculate the new position P' with the use of value x and y

$$x = r \sin \psi$$
$$y = r \left(1 - \cos \psi\right).$$

Simulation is carried out in small but discrete steps, in our case between 15 and 40 milliseconds. This permits us to treat each step as a separate steady turn with a fixed radius and the resulting sequence can create even a more complicated trajectory.

When cornering any vehicle, both single-track and dual-track, a steady turn of a given radius (true circular motion) is never entered and exited instantly, but gradually. Such a curve is called a track transition curve and its ideal shape has been calculated for both road and rail design and is used for their construction throughout the world. For dual-track road vehicles, it is derived from the expectation for steady steering wheel turning – so that the driver can rotate the steering wheel at the same speed throughout the whole transition curve. This matches the clothoid, or otherwise called Euler spiral. Due to lower speeds of bicycles and their greater flexibility in turning due to low mass, and probably most importantly because of the lack of real engineering need[1]), no such curve has been

[1]) Surely, velodrome construction needs very careful design, but the designs are not openly available. Schuermann Architects are probably the best known velodrome constructors.

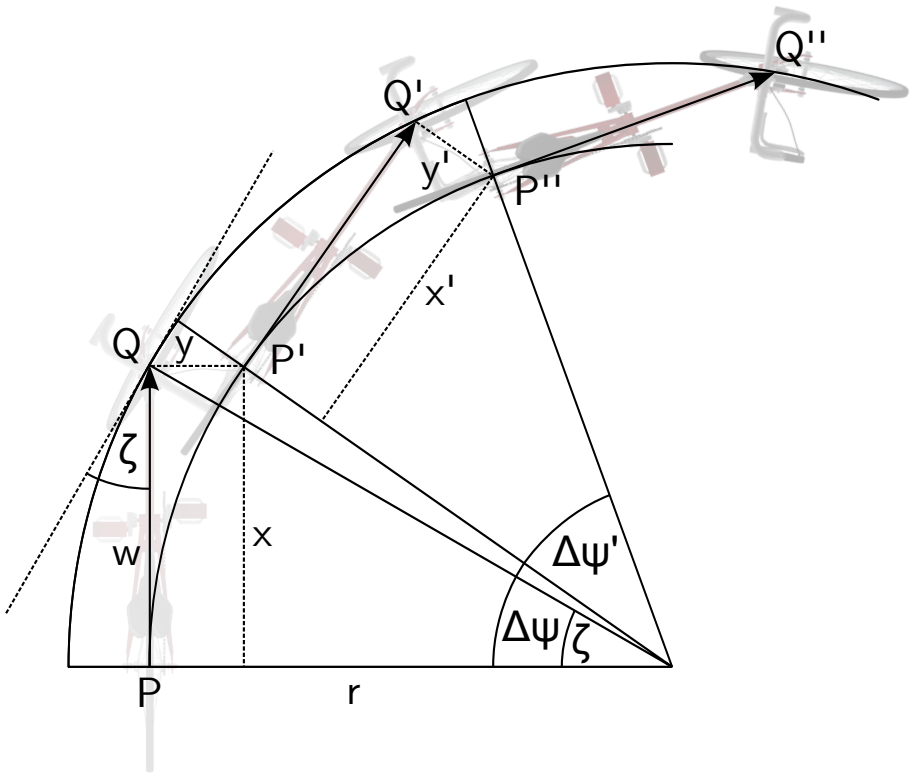


Figure 2.3. Path of each wheel in a turn

commissioned for bicycles. A derivation for bicycles is offered in [4] and instead of steady steering speed, steady leaning speed is used as a premise for calculating the curve. Steady steering cannot be used with single track vehicles, because steering angles are very small, e.g. with a wheelbase of 1m, riding through a turn with a 25m radius only requires a 2° handlebar inclination, whereas the lean at 40km/h will already be 30°.

2.3.2 Lean

During circular motion, centripetal force acts on the object with the magnitude of

$$F_c = \frac{mv^2}{r}$$

where m is the object's mass, v its tangential speed and r the radius of curvature. Apart from that, gravitational force has its effect as we would expect

$$F_g = mg.$$

These two forces have to be in equilibrium for the cyclist to maintain stability. This is attained with lean as illustrated in figure 2.4.

This can be characterised by the following equation

$$\tan\phi = \frac{mv^2}{mgr} = \frac{v^2}{gr}$$

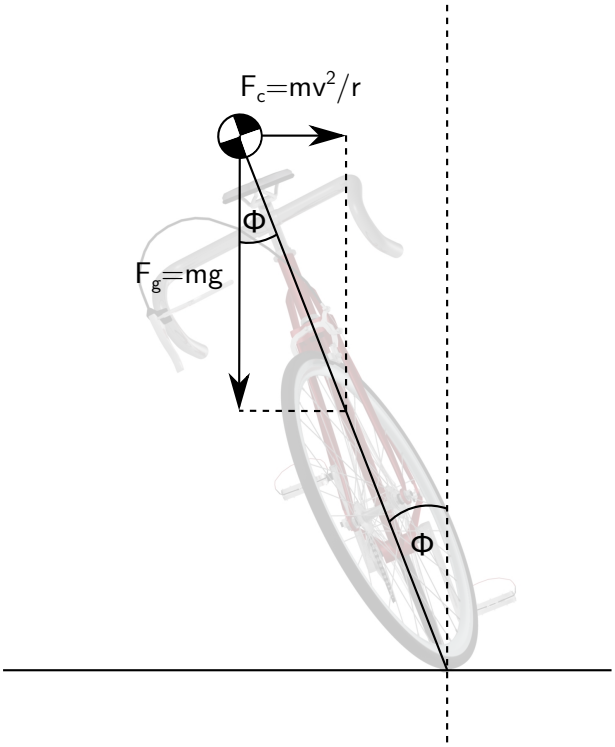


Figure 2.4. Forces acting on bicycle in a turn.

therefore the lean angle can be expressed by

$$\phi = \arctan\left(\frac{v^2}{gr}\right)$$

and is independent of both mass and the height of the centre of mass.

2.3.3 Countersteering

Most bicycle riders acquire their riding skills as children with no theoretical background. Because the rider represents much more mass compared to the bicycle itself, controlling the ride is much easier and sometimes even subtle body movements can affect bicycle heading. Because no great forces are required of the rider to steer and balance the bike, most of these movements are performed with very little conscience and without the rider knowing what is happening in terms of bicycle physics. Hence not many riders know about countersteering though they perform it before nearly every turn.

Conversely, motorcyclists are required to obtain a driving licence in most countries and before receiving it they are subject to both theoretical and practical preparation. Motorcycles are substantially heavier than bicycles and, in most cases, heavier than their riders. On a bicycle it is possible to initiate a turn by leaning the rider's body weight to one side and thus shifting the centre of mass and making the bicycle self-correct the irregularity by turning the handlebars into the direction of lean. (This can be easily observed when riding "no hands".) Due to the different mass distribution on a motorcycle, it is not that easy to shift the centre of mass and that is when countersteering comes in.

Countersteering enables us to initiate the lean even with a heavy motorcycle. The theory of countersteering is very simple: first, before turning in the desired direction, say, right,

the rider briefly turns the handlebars towards the opposite – left. Due to momentum and tyre friction (the front tyre steers the bicycle left, while momentum forces it to carry on in the direction of original heading), a lean in the desired direction is created and finally the rider can turn the handlebars to stabilise the motion for a steady turn.

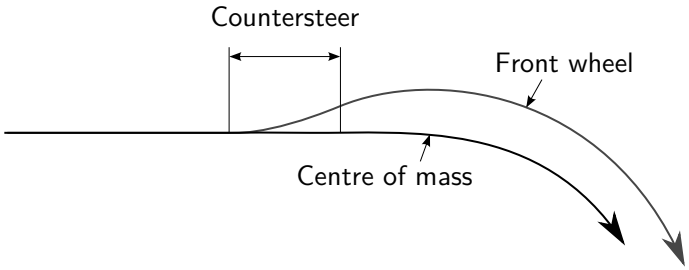


Figure 2.5. Paths of the centre of mass and the front wheel when countersteering.

This phenomena is not apparent to riders, because it happens in a very short amount of time. Yet an interesting situation is given in [3] to prove that riders are unconsciously aware of it: if riding very close to a kerb, the rider feels somehow trapped because he senses that if he rides any closer, he will not be able to turn away from it – that he has no room for countersteering towards the kerb.

2.4 Sideslip

When riding through a turn, the centripetal force that acts on the bicycle must be compensated by tyre friction in order for the bicycle to maintain its path.

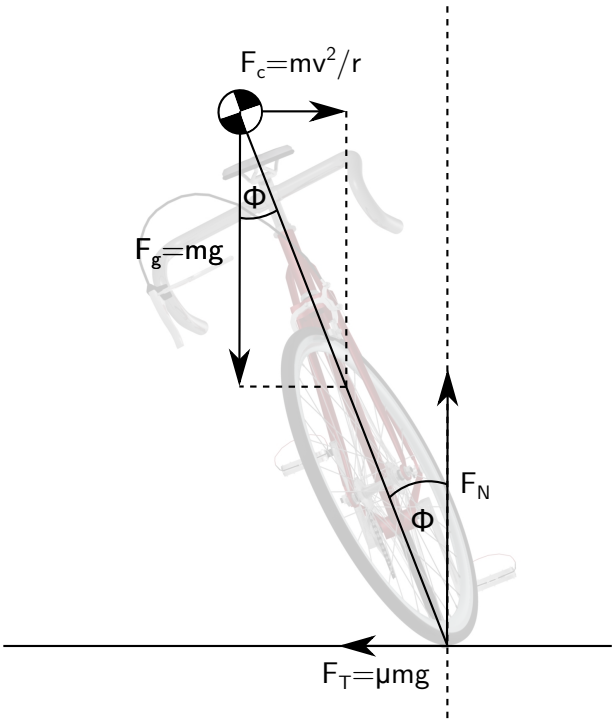


Figure 2.6. Distribution of forces in a turn with tyre friction.

If the centripetal force exceeds the adhesive force the wheel starts slipping and this almost inadvertently results in a fall. Motorcycles have a limited ability to recover from sideslip due to wider tyres. Our interest lies in determining the maximum speed $v_{\max}$ the rider can ride through a turn of a given radius r and slip coefficient μ. Any speed lower than $v_{\max}$ can be considered as a *safe* speed. The maximum speed will occur, when the forces F_c and F_t (figure 2.6) are equal

$$\frac{mv_{\max}^2}{r} = \mu mg$$

therefore

$$v_{\max} = \sqrt{\mu r g}.$$

2.4.1 Banked Turn

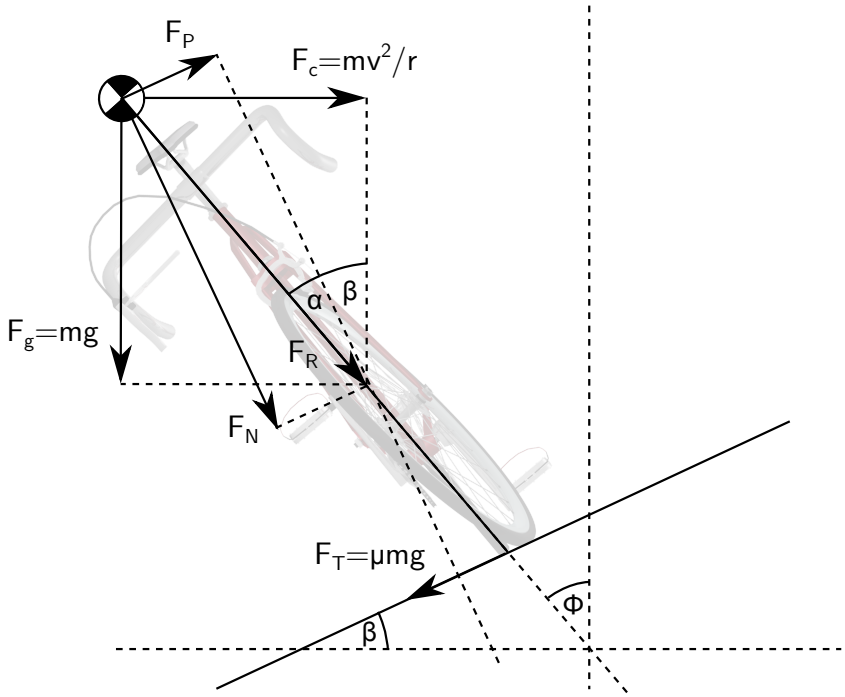


Figure 2.7. Distribution of forces in a banked turn

Banked bends are built to enable vehicles drive through them at higher speeds safely without slipping. While the banking of road bends is quite minor, the banking used for velodromes (venues for track cycling) can be between 30° and 50° [4], mainly depending on their size and therefore the radius of the turn.

Figure 2.7 shows the distribution of forces when riding a bicycle through a banked turn. There are three situations that can occur in a banked turn:

1. The bicycle's lean is perpendicular to the track ($\phi = \beta$). In that case both forces F_R and F_N are equal in size and direction and there is no sideways force making the bicycle slip – as if it was going straight on flat ground.
2. The bicycle is going faster than in case 1. and its lean is greater than the banking ($\phi > \beta$, $\alpha = \beta + \phi$). This situation is depicted in figure 2.7.
3. The bicycle is going slower than in case 1. and its lean is smaller than the banking ($\phi < \beta$, $\alpha = \beta - \phi$). This is analogous to figure 2.7, the directions of the forces F_T and F_P are opposite.

To prevent slipping, F_T has to be greater or equal than F_P. Unlike in the case of flat ground, where it was sufficient only to determine a maximum safe speed, we also have to determine a minimum safe speed – the banking can be so steep that a slowly moving bicycle slips to the inside of the turn. [4] offers a way to derive these speeds which I present here.

The adhesive force F_T can be calculated as

$$F_T = \mu F_N = \mu F_R \cos\alpha$$

and the parallel force F_P (as in parallel to F_T after force decomposition of F_R)

$$F_P = R \sin\alpha.$$

Because

$$F_T \geq F_P$$

must hold, we can write

$$\begin{aligned} F_R \sin\alpha &\leq \mu F_R \cos\alpha \\ \tan\alpha &\leq \mu. \end{aligned}$$

The ratio between F_C and F_g can be expressed as

$$\begin{aligned} \frac{F_C}{F_g} &= \tan\phi = \tan(\beta+\alpha) \\ &= \tan(\beta-\alpha) \end{aligned}$$

for cases 2. and 3. respectively, therefore

$$\begin{aligned} \frac{\frac{mv^2}{r}}{mg} = \frac{v^2}{rg} &= \tan(\beta+\alpha) \\ &= \tan(\beta-\alpha) \end{aligned}$$

so the square of speed can be written as

$$\begin{aligned} v^2 = \tan(\beta+\alpha) &= rg\frac{\tan\beta - \tan\alpha}{1+\tan\beta\tan\alpha} \\ = \tan(\beta-\alpha) &= rg\frac{\tan\beta + \tan\alpha}{1-\tan\beta\tan\alpha} \end{aligned}.$$

Because $\tan\alpha \leq \mu$ must hold, the final equations for the maximum and minimum velocities in a turn with a banking angle β are

$$\begin{aligned} v_{\max} &= \sqrt{rg\frac{\tan\beta+\mu}{1-\mu\tan\beta}} \\ v_{\min} &= \sqrt{rg\frac{\tan\beta-\mu}{1+\mu\tan\beta}}. \end{aligned}$$

2.4.2 Coefficients of Friction

Slip has always been dependant on μ, the coefficient of friction. These are obtained empirically in laboratory tests and depend on the quality of the tyre, road surface and other environmental factors such as the surface being wet. [3] offers common values for different conditions, these are presented in table 2.3.

Surface	μ
Concrete or asphalt (dry)	0.8-0.9
Concrete or asphalt (wet)	0.4-0.7
Gravel	0.6-0.7
Sand	0.3-0.4
Icy	0.1-0.2

Table 2.3. Coefficients of friction for various road surfaces and conditions.

2.5 Resistance

Like all other vehicles, the force propelling the vehicle is resisted by a number of factors. The ones that have to be taken into account in bicycling are:

1. The resistance of bearings and the drivetrain
2. Rolling resistance of tyres
3. Sloped terrain
4. Air drag

2.5.1 Drivetrain and Bearings

[4] states that the efficiency of the bicycle's drivetrain is 96%, where 2% comes from bearing resistance and the other 2% from the chain itself. Such values are too small to be noticeable by the user within the simulation, but can easily be included as a constant C_B biasing the propulsion force.

2.5.2 Rolling Resistance

Similarly, rolling resistance of a well inflated tyre on a good surface has a relatively small influence on the ride, though it can be very well noticed by a rider on a mountain bike with wider tyres when riding off road. Rolling resistance is defined as

$$R_r = mgC_{rr}$$

where C_{rr} is the rolling resistance coefficient. [4] uses a rolling resistance coefficient for a well inflated tyre and a concrete surface between 0.005 and 0.01.

2.5.3 Sloped Terrain

We can interpret riding on sloped terrain as a force either acting against propulsion (when riding uphill) or contributing to it (downhill). The force is computed as

$$R_s = mg \sin \alpha$$

where α is the slope of the terrain.

2.5.4 Air Drag

Air drag poses the biggest obstruction for cyclists to reach high speeds, as the force increases quadratically in respect to speed. Air drag is defined as

$$R_D = \frac{1}{2}\rho v^2 C_D S$$

Machine and rider	*Drag coefficient C_D*	*Frontal area S, m^2*
Upright commuting bike	1.15	0.55
Road bike, touring position	1.0	0.40
Racing bike, rider crouched, tight clothing	0.88	0.36

Table 2.4. Drag coefficients and frontal area in selected situations

where ρ is the density of the environment (air), C_D the drag coefficient, mainly dependant on shape, and S the cross-sectional area.

[3] offers a table of common values for the drag coefficient C_D and the frontal area S. A selection is presented in table 2.4. We will choose average values for the simulation, that of a road bike with the rider in touring position.

Air density ρ is defined as

$$\rho = \frac{p}{R_{\text{specific}} T}$$

where p is air pressure, R_{specific} the specific gas constant for air, 296.96J·kg^{-1}·K^{-1} [3], and T the temperature in Kelvins. [3] states sea level air density at 20°C to be around 1.2kg·m^{-3} which is sufficient for our needs. But because air drag is the major resistive force on the bicycle, it is important to note that a major change in altitude can have a considerable effect – an example is shown in [4], where at 200m above sea level a cyclist reaches a speed of 48.6km·h^{-1} at a given power output; at 2000m above sea level, with the same power output, he can reach 54.7km·h^{-1}.

2.6 Acceleration and Braking

Acceleration and braking of a bicycle follows the rules of classical mechanics. Additional factors that have to be included are resistances (explained in the previous section 2.5) and the physical properties of the rider which will be presented in more detail in section 2.7. For the purpose of determining acceleration, it is only important to know, that the rider will output a certain amount of power P at any given time.

For the simulation, all longitudinal forces, acceleration and speed are used as scalar values. Their direction is derived from the state of the bicycle (section 2.1); a new state is computed in each timestep, based on the previous state and interpreting user input and collision detection. This will be explained in more detail in chapter 6.

2.6.1 Acceleration

Acceleration a is defined as

$$a = \frac{\Delta v}{\Delta t}.$$

This will be used to determine the change in speed in every simulation timestep, over which the value of acceleration remains constant. It is computed from the longitudinal forces acting on the bicycle and rider.

The only force that has not been mentioned yet is the propulsion force – the power of the rider transmitted into a force acting on the rear wheel to propel the bicycle. This is

$$F_P = \frac{P}{v} C_B$$

where P is the current power output of the rider and v the current speed. C_B is the drivetrain and bearing loss constant of 0.96 as described in section 2.5.1.

Acceleration is calculated in respect of force as

$$a = \frac{F}{m}$$

which, when we combine the previous sections, gives us a final equation for acceleration

$$\begin{aligned} a &= \frac{F_P - R_r - R_D - R_s}{m} \\ &= \frac{\frac{P}{v}C_B - mgC_{rr} - \frac{1}{2}\rho v^2 C_D S - mg\sin\alpha}{m} \end{aligned}$$

m being the total mass of the bicycle and rider.

2.6.2 Braking

Braking can be seen as another resistive force. Simulating it perfectly would require to take into account various properties of the braking mechanism which can have a large effect in the real world (e.g. the feel of hydraulic disk brakes on a mountain bike opposed to calliper brakes with worn pads on an old road bike) but would go unnoticed in the computer simulation. [3] have come with an approximate maximum value of deceleration attainable on a conventional bike to be 0.56 $g = 5.45\text{m}\cdot\text{s}^{-2}$. As brake force is applied, the weight distribution shifts towards the front wheel, at first making the rear wheel skid if also braking and finally making the bicycle longitudinally capsize throwing the rider "over the bars". The aforementioned value is an approximate limit before this happens.

2.7 Physiological Properties of Rider

The human body is a complex mechanism which cannot be easily parametrised. Apart from that, the fact that every human being is unique contributes to this complexity. [3] presents a detailed insight into research conducted on human power generation so far. This mainly consists of series of experiments on riders with the goal to asses sustainable power output for varying time spans. Backing medical research explains how energy is handled within the human body.

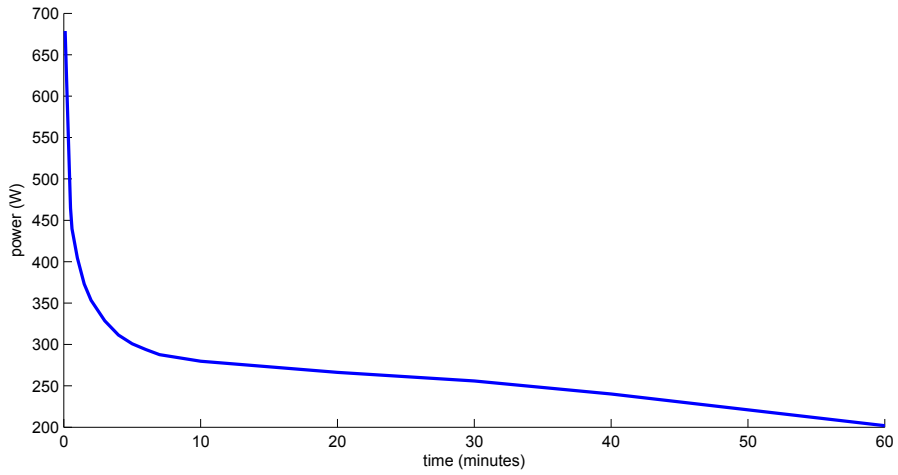


Figure 2.8. NASA power-duration curve for "healthy men" [6]

Apart from being able to produce a steady low power output across long amounts of time (minutes – hours), the human body is capable of producing substantially more power for a short period (seconds). Called anaerobic power, it is achieved by "rapid partial metabolism of glycogen without oxygen" [3]. Rider fatigue in the simulator follows the principles of this process, because long term fatigue would be impossible to simulate as riders are capable of riding long distances at moderate speeds with relative ease without the fatigue being apparent on the power output. I do not expect the simulator to be run hours on end. Fatigue is parametrised by a value between 0 and 1, 1 meaning "exhausted" and set to recover within 60s. Fatigue is incremented whenever the user makes the cyclist use more than 50% of available power. In retrospect, fatigue biases the value of maximum available power to loosely match power-duration curves presented in [3], specifically the NASA curve for "healthy men" originating from [6], reinterpreted in figure 2.8. In the simulator, three power values are used: P_{max}, P_{med} and P_{min}. Maximum available power is then computed as

$$P_m = \begin{cases} 2\mathrm{f}\,(P_{med} - P_{max}) + P_{max} & \text{if } \mathrm{f} \in [0, 1) \\ 2\left(\mathrm{f} - \frac{1}{2}\right)(P_{min} - P_{med}) + P_{med} & \text{if } \mathrm{f} \in [0.5, 1]. \end{cases}$$

where f denotes fatigue. To reflect the NASA curve, I set P_{max} to 800W, P_{med} to 400W and P_{min} to 200W.

Fatigue is indicated by the colour of the letter F in the lower right corner of the screen, green meaning fresh and red meaning exhausted.

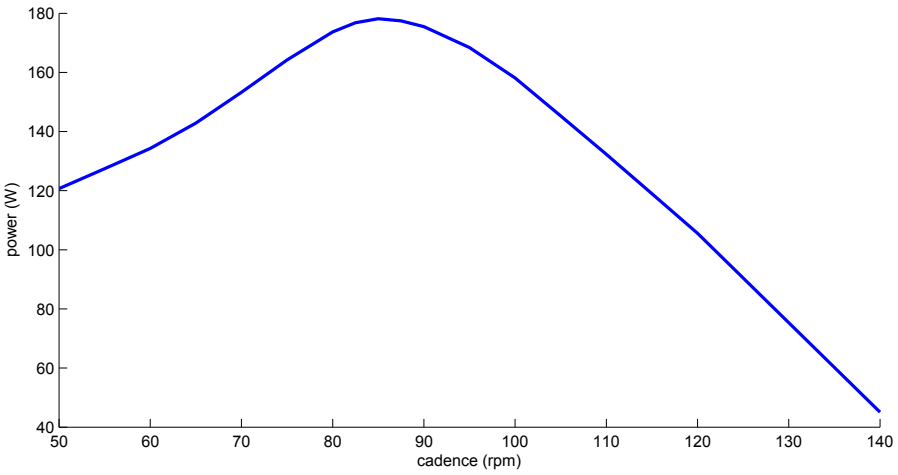


Figure 2.9. Effect of cadence on pedalling power output [7]

The last factor affecting maximum human pedalling power output is pedalling frequency. Although not explicitly referred to in a whole section in [3], it is frequently mentioned and its effect on power is charted in a graph illustrating a different topic – backward vs. forward pedalling. The curve for forward pedalling taken from [7] is shown in figure 2.9.

I chose to approximate this curve with the cosine function as follows:

$$P = \begin{cases} P_m(\frac{1}{2} + \frac{1}{4}(-\cos(\frac{f}{85}\pi) + 1) & \text{if } f \in [0, 85) \\ P_m\frac{1}{2}(-\cos(\frac{f}{170}2\pi) + 1) & \text{if } f \in [85, 170]. \end{cases}$$

where f is cadence (pedalling frequency in rpm). The value of P is then the maximum power output available to the user in each simulation timestep and the controller position determines what proportion of the power will be used for propulsion.

The effect of pedalling frequency on power output is the reason why nowadays gearing is used on a large proportion of bicycles. The simulator offers a limited choice of gears, roughly resembling a Sturmey Archer S3X[1]) internally geared, 3 speed, fixed gear hub. Gears can be changed by pressing the buttons 1, 2 and 3 on the keyboard. Nevertheless, the simulator is ready for a much wider gear range that could be implemented with only minor changes.

[1]) `http://www.sturmey-archer.com/products/hubs/cid/3/id/47`

Chapter 3
User Input

In a simulator, the user input method plays an important role in making the simulation seem realistic. Fortunately, controllers nowadays have a somewhat unified interface, only leaving the programmer with a certain number of control axes and their current values, irrespective of where the values came from, be it a joystick, steering wheel or a specially designed single-purpose device. This is a great advantage as it facilitates portability and makes any compatible controller usable.

Because of its wide availability, I chose the joystick to be the controller of choice to test the simulation on. However, throughout the testing I concluded that this choice was not ideal as a joystick does not offer enough precision and altering the controller by very small steps is very difficult. Although it is possible that a higher quality joystick would offer better control, I chose to focus on the steering wheel, as it offered much higher precision. Nevertheless, I also present my findings when trying to set up the simulation with a joystick. The simulator supports 3 modes of controller operation, differing in sensitivity and handlebar threshold – steering wheel mode, 4-axis joystick mode and 2-axis joystick mode.

3.1 The Joystick

Because the main use of modern joysticks is flight simulation, they commonly come with 4 axes and a series of buttons. In flight simulators, the forward/back axis of a joystick controls the aircraft's elevators to regulate the pitch, the left/right axis controls the ailerons that govern roll, turning the stick around its vertical axis controls the rudder for yaw (pedals are used in a real aircraft) and lastly a slider sets throttle.

For the purpose of simulating a bicycle, the controls have to be remapped. It is quite evident, that the forward and back axis would represent the propulsion and braking force respectively.

When it comes to turning and leaning, the situation becomes more complex. At first I thought of using the main left/right axis and the vertical axis for a separate control of turn, lean and their effect on stability, the amount of countersteering etc. After a more in-depth study of bicycle physics I concluded that this would be directly *against* the principles the bicycle works on. To retain stability, the amount of turn and lean has to be *exactly* right just as the physics predict, their values being constantly biased by tiny amounts to keep the bicycle upright. All of this is done without the rider consciously noticing that these microoperations are taking place and it would be *unnatural* to force the user to pay attention to this.

In this respect we expect the simulated rider to be a rider experienced enough to be able to govern these operations automatically (as basically any person able to ride a bike is). This leaves us with an opportunity to use two axes for steering. As mentioned in section 2.3 about turning, handlebar turn angles are usually very small during normal riding which induces limitations concerning controller precision. Thus these two axes can be used as major and minor control, each having different sensitivity and threshold.

Finally, we are left with the slider that controls throttle in flight simulators. In the original concept, I thought of this control element to govern forward and backward lean, making it e.g. possible to jump on and off kerbs. This though is out of the scope of this thesis and so the slider can serve as a hardware sensitivity and threshold controller, making it possible for the user to set the sensitivity to suit their feel and adapt it to the track being ridden through.

3.2 The Steering Wheel

After finding out that most gaming steering wheels offer an option to set degrees of rotation, the steering wheel quickly preceded the joystick in suitability for the task. When degrees of rotation are set to 900° (meaning that the controller makes 2.5 revolutions between reporting peak values, similarly to a standard car) and handlebar threshold is set to be 45° (a cyclist rarely exceeds such an angle), a 10° turn of the steering wheel initiates a 1° turn of the handlebars. Such precision offers substantially more natural control without the somewhat obscure setting required by the joystick.

3.3 Critical Situations

There would not be much reason for a simulator if one could ride any way they wanted without the risk of crashing or falling. In real life, with training, a rider can predict and prevent certain critical situations by adapting their riding style to the conditions (e.g. sharp turns, slippery road). This simulation can not enable the user to feel the situation as he would in reality, hence they must receive an alert for the coming situation – the danger of slipping and falling. It is indicated by a letter L of varying colour from green to red in the bottom right corner of the simulator screen.

3.4 Gradual Leaning

The user is capable of turning the controller at a similar rate as a cyclist would turn the handlebars. In real life, quick turning can only occur at low speeds – but at higher speeds initiating a turn requires more technique and time (see section 2.5 on countersteering). The simulator has to respect this phenomena and through a set of parameters controls the way change in controller input is evaluated. These include maximum lean acceleration, set to 150°s^{-2}, and maximum lean speed, set to 50°s^{-1}. Lean speed is recorded in the previous timestep and cannot change more than acceleration allows. Stopping at a desired lean angle on the other hand is performed faster, on a fixed lean angle difference $\Delta\phi$ =5°. Deceleration is calculated so that the simulator handlebar angle does not exceed the desired handlebar angle set by the controller – otherwise it would create oscillations. Its value is calculated as

$$\alpha = \frac{\omega^2}{2\Delta\phi}$$

where ω denotes current lean speed. It might seem that deceleration should be similar to acceleration. In fact it is not the case in real life – it is easier and faster to stop the lean at a point of equilibrium than to exit it.

3.5 Proposed Ideal Controller

To evoke the feeling of riding a bicycle in a simulation, the most logical way to do this is to use a modified real bicycle as the controller as has previously been done on the Tacx trainer or the FIVIS simulator, both mentioned in section 1.2. Any implementation requires a great deal of hardware work, but can be designed in a way to communicate as any other human interface device attached to the computer, thus not needing any changes to the software, except for calibration. Both the handlebars for steering input and a lever (levers) for braking should not require much engineering work and financial expense. Greater attention has to be paid to the precision of the controller, as the angles handlebars are commonly turned to are very small and too little precision would not lead to a good experience. In contrary, the propulsion mechanism is rather complicated, because it requires a device to apply variable resistance to the pedalling rider. This would also require modifications to the physics implementation, as the physics engine would not do the calculations of acceleration or speed – in turn it would only receive a value for speed taken from the actual spinning of the rear wheel. The "calculation" would thus take place in real life, with the mechanism applying the resistive forces to the rear wheel making pedalling harder for the rider.

Chapter 4
3D Modelling and Animation

4.1 The Bicycle

As my semestral assignment for the Multimedia Applications course (A7B39KMA) I worked on a 3D model of a bicycle. It was intended for static rendering and therefore didn't pose limitations on the level of detail. Beside polygons, NURBS curves and surfaces were dominantly used in many parts of the model so the original model itself proved unsuitable for the purpose of the simulator. Nonetheless, it provided a good starting point for a conversion to suit real-time rendering.

Figure 4.1. Render of the final polygonal bicycle model used in the simulator

The model is based on my own fixed gear[1]) bicycle which I commonly use for everyday commuting. As I had previous experience with a bicycle, modelling did not pose any great difficulties and I was even able to reuse various parts of the original model or carefully convert NURBS surfaces to polygons. It was important to correctly set rotation pivots to be able to rotate wheels, the fork and of course the whole bicycle in the simulator.

[1]) A fixed gear bicycle is a bicycle with no freewheel, therefore not allowing the rider to coast. Fixed gear bicycles are essentially no different from track bicycles, except for the occasional addition of a brake. They were adopted for urban use by bicycle messengers due to their simplicity, durability and easy maintenance, but have gained much greater popularity in recent years.

For simplicity, I also prepared crank and pedal rotation in Maya using keyframe animation. Apart from prospective easy manipulation in the simulator, I chose this option because I needed a reference to set pedalling animations, explained in section 4.2.2.

4.2 The Rider

Autodesk Mudbox offers great manipulation tools for character modelling. It comes with a prototype human body that can serve as good base mesh. Autodesk Mudbox can be linked with Autodesk Maya and changes made in one program can therefore be instantaneously sent from one program to another which is a very convenient feature. The strength of Autodesk Mudbox lies in its sculpting tools and most importantly texture painting tools. Maya on the other hand offers great tools for precise vertex manipulations within the mesh and then of course all that is required for animation.

Figure 4.2. Human body textured in Autodesk Mudbox, exported into Autodesk Maya with a skeleton prepared for binding

4.2.1 Body and Texturing

I began with the body prototype offered by Mudbox and first made several rough alterations to it in Maya – I duplicated the mesh and separated the parts that resemble clothing – the T-shirt, the shorts, the shoes and the cycling cap[1]) and modified their geometry to match their use. Each part could then be transferred back to Mudbox for texturing.

[1]) Some might object that a cyclist should wear a helmet. The cycling cap, or casquette, has been traditionally connected to the sport and was designed to be utmostly practical for racers. It was only the 2003 Giro d'Italia where helmets first became compulsory in an UCI sanctioned race. Unfortunately, since then, cycling caps have nearly disappeared from professional cycling. I believe I should help keep this piece of attire alive.

At this point I searched the Internet for images suitable for texturing a human body that would be freely available, but with no avail. Therefore I decided to use photographs of myself as reference images for texture painting in Mudbox. It proved to be a good decision as I had control over the whole process and could make new images if the ones I had so far did not fit ideally. When creating the photographs, great attention has to be paid to even lighting – when looking at whole images, the colours and intensities may appear very close, but when painting the textures side by side, the difference can be quite profound.

Texture painting in Mudbox is carried out by painting over a stencil – reference photograph – onto the mesh through the current 3D view. Mudbox automatically creates UV mappings for the meshes and paints the stencil onto an image representing the texture for the mesh. Though relatively straightforward, this process requires great care and precision even to achieve moderate results.

The trickiest part of the process was the face. When comparing the prototype head offered by Mudbox with my own reference photograph, it was evident that the mesh would have to be sculpted for a better fit and better results. For sculpting, Mudbox requires to add levels of detail so the result is smooth, as most organic structures are. The sculpting tools can be compared to playing with Play-Doh but like texturing, they require great care and precision, or even more so. Even subtle alterations can create a completely different facial expression.

4.2.2 Animation

Once finished with texturing, the model, or skin in animation terminology, is ready for its skeleton, binding and finally, animation. [8] offers a concise manual for the tools necessary for creating an animated character.

An animation skeleton is made of bones and generally copies a simplified human skeleton. It is illustrated in both figures 4.2 and 4.3. Once it is bound to a skin, moving the bones deforms the geometry of the original skin to simulate the effect bones have in an organic structure. Each bone influences a cluster of vertices and with smooth binding, each vertex can be influenced by more than one bone. Binding in Maya is done automatically with the Smooth Bind or Interactive Skin Bind tool. Although Smooth Bind works very well, I had to make use of the Interactive Skin Bind tool which offers comfortable bone influence editing. The reason for this was arm orientation on the skin – instead of having arms stretched out sideways, they were only slightly stretched diagonally and too close to the rest of the body and so the Humerus also influenced a large proportion of the chest. Why the prototype human body in Mudbox is so designed I cannot say, as when I later researched, virtually all characters are set up with their hands stretched sideways.

Once bound, the skeleton and skin are ready for animation. I use a combination of inverse and forward kinematics. The first pair of IK handles are attached to the legs and account for pedalling – their start joint being the hips and the end-effector the ankles. These IK handles are attached to motion paths (NURBS circles) and set to revolve around the circle within the timespan of 0-1s to match the cranks. After that, the orientation of the ankles is fine tuned with the use of forward kinematics and keyframed. In addition to that, within each crank revolution, the rest of the body is slightly swayed from one side to another and back as a rider does when pedalling. To make it simple, the joint between the arms remains stationary (which is not far from real life, where the upper body does very little movement, if any, during pedalling). This enables us to create an independent arm animation for handlebar turning. Motion between the shoulders and wrists is governed by a second pair of IK handles effecting on the wrists; these are attached to Bézier animation curves following the handlebars. Hands are then once again set up by forward kinematics to

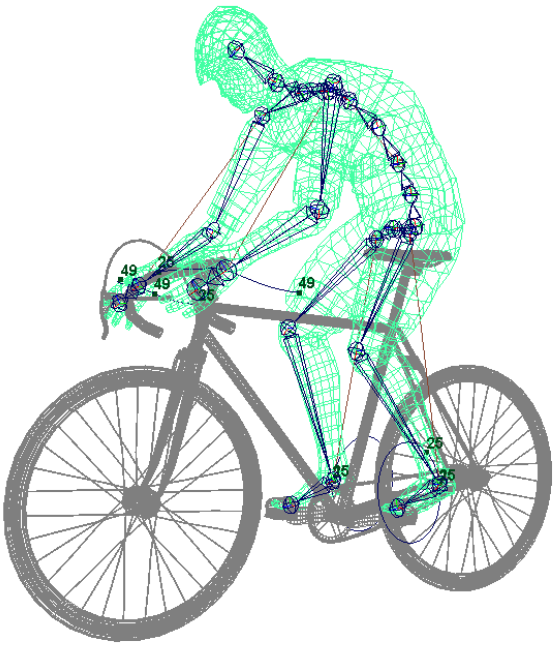


Figure 4.3. Skeleton used for animation together with animation curves and IK handles

match the bars. Fingers are not animated, as they would require approximately the same number of bones as the rest of the skeleton, which could pose a performance issue. Finally, on export to the FBX format, animations have to be baked so that the FBX SDK can interpret the information correctly. This is discussed in detail in chapter 5.

4.3 Tracks

The last modelling task are the three tracks: a velodrome, a road with bends and offroad terrain. Once again, Autodesk Maya is the program of choice to create these. All textures used in the tracks come from CGTextures[1]) whose licence allows them to be used in 3D graphics applications and to be bundled within. Their licence is available at `http://www.cgtextures.com/content.php?action=license`.

4.3.1 The Velodrome

A velodrome is the venue for track cycling. It consists of two 180-degree banked turns connecting to two straights. Though hundreds have been built around the world, virtually no publicly available detailed documentation exists apart from information on track length and banking, which is not sufficient for recreating one in virtual reality. I believe velodrome architects have no real need for publishing their track designs to avoid aiding competition.

Characterising the geometry is not at all simple as it might seem when first looking at a track – the turns are not just fixed radius circles but also include transitional curves on entry and exit. An article [9] on the 2012 Olympics track in London even suggests the entry and exit curves to differ to provide the rider an even more fluent transition. Only Chris Nadovich, the engineer behind the 1996 Olympics track in Atlanta openly states the use of Cornu spirals[2]) for transitions in his web presentation [10] about the track. The most

[1]) `http://www.cgtextures.com`
[2]) Cornu spirals are also referred to as Euler spirals or clothoids

detailed information I found was a partial scan of the original design plans of the track in Brno, posted on the Fixed Beer Brno blog[1]), which I used as the basis for modelling.

The plans reveal the track to have 400m in circumference, banking of 7° in the straights and 33°20' in the turns, turns of a radius of 30m covering 120° on each side, a transitional curve 34m in length leaving approximately 79m for the straights. The track is 8.3m wide on the straights and 7.8m wide in the turns. I used this information to create a Matlab script to generate a set of curves that could then be used in Maya to loft a surface. The two lofting curves each consisted of three parts, the first representing the straight, the middle representing the transition curve in the form of a Cornu spiral and finally the third representing a circle with a fixed radius. An additional third curve was also constructed to provide height information for the outer curve (the inner curve remains at height 0). I plotted these in Matlab, exported as EPS graphics, edited in Inkscape to remove unnecessary residuals such as the background from the plot and finally imported into Maya using the Create Adobe Illustrator Object tool. In Maya, I aligned the curves, smoothed them and finally constructed the surface of one quarter of the track with the Loft Surface tool. Before duplicating the quadrants to make a complete track, I applied textures, as the loft tool produces easily editable and aligned UVs. I created the sidewalls analogously with the addition of the Extrude tool. The scene also contains basic ground textured to represent grass and a skydome is added in the simulator.

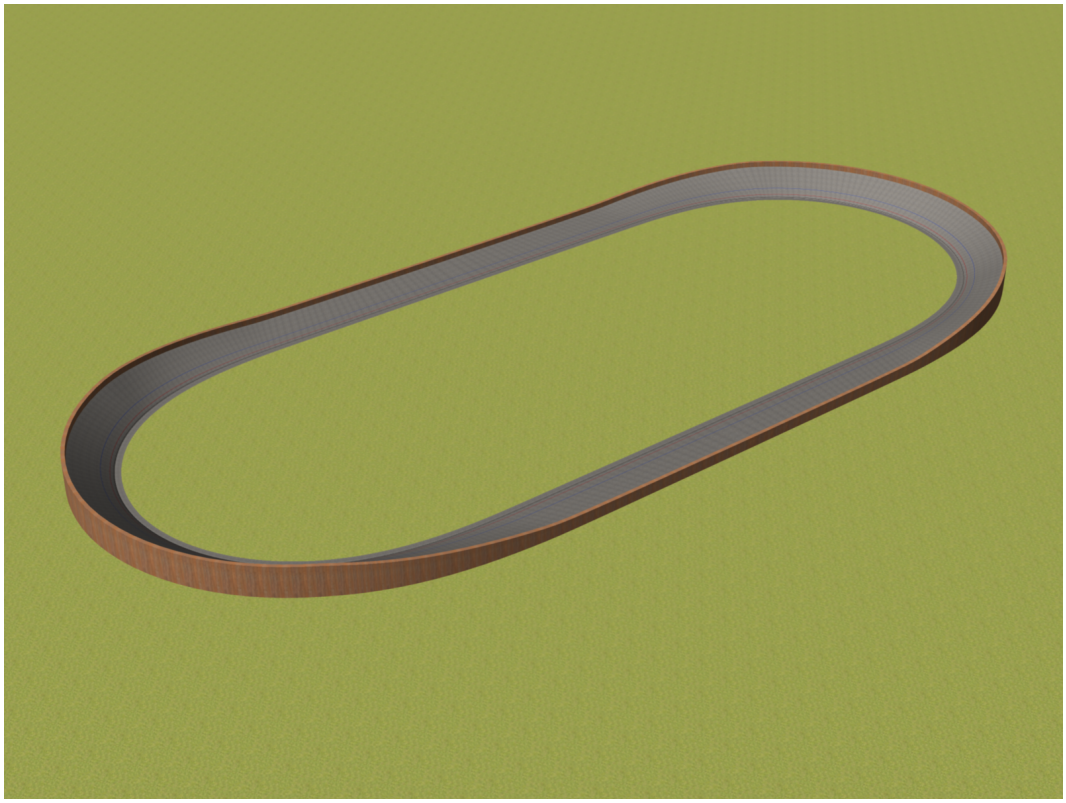


Figure 4.4. A render of the modelled velodrome used in the simulator

Velodrome tracks include three marking lines, each 5cm wide. Beginning with the innermost, the black line is called the measurement line and marks the length of the track and ideal path. Next is the red sprinter's line, commonly marked at 1m from the inner edge of the track: if a cyclist rides below this line others are not allowed to overtake from the left.

[1]) `http://www.fixedbeerbrno.cz/2012/10/vykres-brnenskeho-velodromu.html`

The third line – the stayer's line is marked in blue and is usually placed at one third of the track width, at 2.5m from the inner edge at least. It is used in madison[1]) races by riders not currently competing and resting by riding slowly above the line, not to interfere with the rest of the racers.

The 25cm wide flat patch inside of the track is called the côte d'azur and is not used for racing. Commonly being painted blue, the name originates from the French translation of "blue coast".

4.3.2 Road with Bends

This track resembles a windy mountain road with eleven bends and switchbacks. Such roads at higher altitudes are commonly narrower and so the road width used is approximately 4m. But even this allows the bicycle to turn around comfortably without leaving the asphalt. Some of the bends are banked to aid lean. When riding the track, it is important to pace the cyclist when ascending – riding at full strength will certainly exhaust the cyclist and he could have very little strength in the final climb. It is possible to reach the top with very little fatigue.

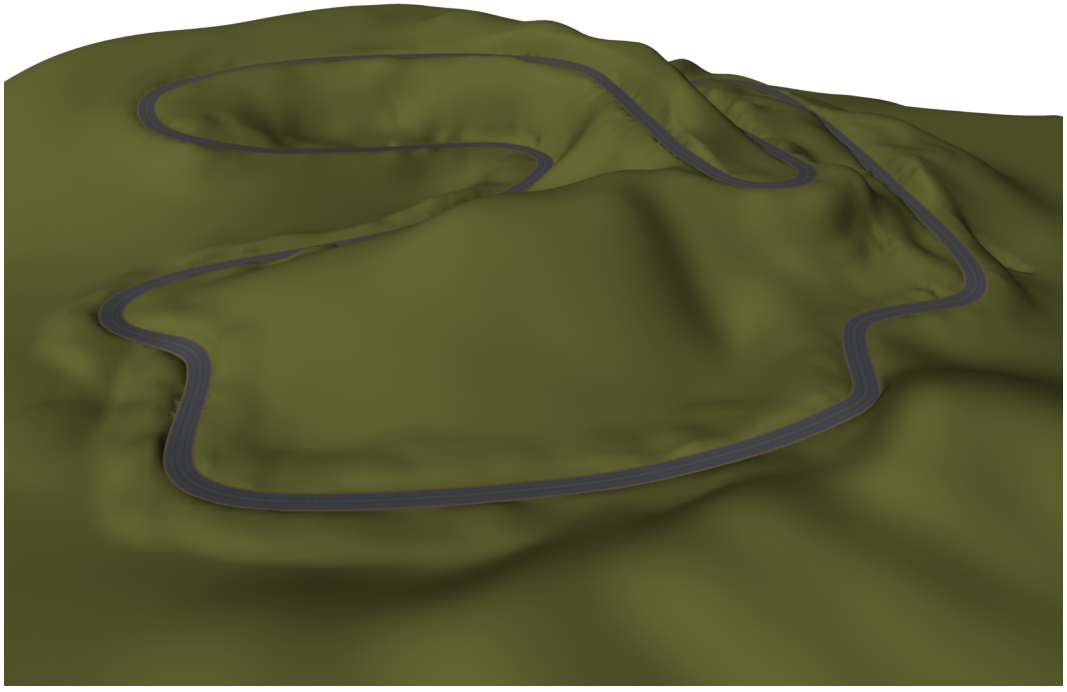


Figure 4.5. A render of the modelled road

To start I sketched myself the track on paper and after a few revisions I started modelling. I used the CV Curve tool in Maya to draw a Bézier curve in top view to create the layout. After that, I modified the curve control points to create the track's elevation. I duplicated the curve and moved the duplicated curve up slightly and lofted a polygon surface between them. The reason for lofting vertically is the fact that this allowed me to use the Extrude tool to horizontally extrude the surface's lower edge loop on either side. After deleting the

[1]) A madison is a team race roughly comparable to a relay race. It is named after the Madison Square Garden in New York, where such races were first held.

original loft, this left me with a 2-polygon wide track with the original curve exactly in the middle. I could then still edit irregularities of the lofted surface by modifying the original curve as Maya keeps these connections before they are explicitly deleted. The road could be easily symmetrically textured because extruding on either side left the UVs mirrored.

The next part was the surrounding terrain. I decided to sculpt the terrain in Mudbox because it offers much more powerful tools for the task. Though the terrain looked very nice and even relatively realistic, it was near to impossible to be precise enough to match the track so that it does not float in mid-air. Therefore I took on the lengthy process of aligning the vertices of the terrain's mesh with the track, one by one. While this produced good results in positioning, it broke vertex normals where the terrain next to the track was steeper. I attempted to fix the most affected parts but with the relatively high number of vertices, the Vertex Normal Edit Tool in Maya was very difficult to use effectively. Though not visually perfect, the track still serves its purpose of demonstrating the simulator's operation.

4.3.3 Offroad Terrain

Offroad terrain consists of a number of hills covered in grass. I sculpted the landscape in Mudbox and added a few trees and stones to better demonstrate collisions. I downloaded the models of the trees from the Loopix-Project[1]) and the stones from Archive3D[2]).

[1]) `http://www.loopix-project.com/`
[2]) `http://archive3d.net`

Chapter 5
3D Model Loading

Even though retaining geometry data is crucial for virtually all computer graphics applications, no universal format exists for the purpose. In contrary to 2D images, the support of different formats significantly varies from one application to another and many studios use their in-house developed formats that suit their specific needs.

[11] offers a comparison of available file formats including FBX, COLLADA, MD5 and OBJ. Both FBX and COLLADA are the major competing file formats mainly used for geometry data exchange between applications and are both backed by extensive frameworks. This has both its upsides and downsides – the formats are complex in order to offer all sorts of functionality 3D applications might need and, with their primary goal being data exchange, high performance cannot be expected. On the other hand, the framework supplies the programmer with built in functions to manipulate with the data with ease. In contrary, the OBJ format is very simple and does not require such a large framework, but can only store geometry and material data.

Apart from obviously being able to hold geometry data, I required the format to support skinning and skeletal animation, which both FBX and COLLADA meet. The FBX format is developed by Autodesk Inc. and is the standard interchange format between most Autodesk's 3D applications. Although plugins exist for exporting the COLLADA format from Autodesk Maya, I chose to stay with the FBX format hoping for flawless conversion and manipulation which I can so far confirm.

5.1 The FBX Format

The FBX format originated as the new native format for Kaydara's Filmbox software in 1996 – an application used for motion capture and character animation. In 2002, Filmbox was renamed to MotionBuilder and after a series of acquisitions it became part of Autodesk Inc. in 2006. Autodesk have since adopted FBX to be the native interchange format between their portfolio of 3D applications. For example, when Autodesk Mudbox and Maya are linked as discussed in section 4.2, data is transferred automatically between the applications via the FBX format.

Autodesk offers a large software development kit to provide programmers with comfortable access to all the features the format has to offer. Although the SDK is well documented [12], due to many major changes which came with the 2011 release, mainly concerning the animation system, searching for other online support in forums was not very effective, as many posts discussed the use of the pre-2011 API. My project uses the 2013 FBX SDK and does not use any of the deprecated functions.

The FBX format is binary only and in order to extract scene information, it has to be loaded into memory as a whole. This process though is governed by the `FbxManager` and `FbxImporter` classes in the SDK and does not require special attention. Once in memory, an FBX scene (an instance of the `FbxScene` class) resembles a scene graph structure similar to most computer graphics applications. It is of course identical to how the modeller left it on export from the 3D modelling application. Nodes in the scene graph are instances of the

FbxNode class and offer access to all data concerning the node, be it its type (FbxNodeAttribute) or any of its properties (FbxProperty) such as materials or transformations. If the node is a mesh, it also offers access to geometry data such as vertex positions, indices, normal directions and UV coordinates. If the mesh has a deformer (a bound skeleton), transformation matrices can be accessed for each cluster (the set of vertices a bone influences) that are necessary for correct deformation evaluation as well as a list of indices influenced by the given cluster along with their weights. In order to evaluate animation, an instance of the FbxAnimEvaluator class can compute a transformation matrix for a given node at a given time, which the programmer has to use in conjunction with the cluster transformation matrices to compute the vertex positions at any given time.

Although the FBX SDK offers its own scene graph, in order to be able to render the geometry via OpenGL, I also had to create my own supplementary scene graph to hold the geometry, transform and deform information in a format suitable for passing through the OpenGL graphics pipeline.

5.2 Implemented Features

FBX manipulation is implemented in a series of classes: FbxHandler, FbxSceneNode and FbxMeshNode that can be found in the FbxHandler.cpp file. The FbxHandler class takes care of scene loading and keeps a list of pointers each pointing to a FbxSceneNode representing the root node of each loaded scene. The FbxSceneNode is a base class that contains variables to hold information common for all nodes in a scene graph, such as transformations, and virtual functions such as Render() to be extended in a derived class. The FbxMeshNode class is derived from FbxSceneNode and adds functionality for mesh manipulation such as geometry data loading and storing. In the case of a mesh with deformation, it also extracts cluster information to be later used in deformation evaluation. The evaluation itself takes place in the vertex shader and will be explained in section 7.4.1. Each node holds a pointer to its original node in the FBX scene graph to provide easy access to necessary information at any time.

As outlined above, only the features necessary for the simulator are implemented and other node types such as NURBS curves or surfaces, cameras or lights are ignored (or precisely only loaded as a FbxSceneNode with no relevant information). The implemented features include:

- meshes and their material properties including their ambient, diffuse and specular components, together with textures and their UV mapping
- mesh deformations
- animation evaluation for all nodes.

To my knowledge, the only important requirements posed on a FBX exported file to be suitable for proper rendering in the simulator are:

- inclusion of per vertex normals
- a triangulated mesh
- baked animations (if included)
- a maximum of 4 bone influences per vertex.

Except for bone influences, all can be set in the FBX exporter for Maya and I expect similar functionality in exporters from other modelling software. In the case of bone influences, care has to be taken at skin binding, but a maximum of 4 influences is commonly used as default and more are rarely necessary in character animation.

Chapter 6
Physics Engine and Collision Detection

The physics engine is implemented in the `Physics` class defined in `main.cpp`. It incorporates a series of functions to handle all aspects the simulator demands of it. In short, it is speed, acceleration, direction, lean and slip, and lastly collision detection which covers both vertical position and pitch, as well as crashing into an obstacle.

6.1 Dynamics

The heart of the calculation lies in the `Physics::set()` function, which is called in every cycle before rendering a frame. First, time is measured to ensure exact evaluation as timestep length can differ between each frame. Next, new values for controller positions are polled and the `Physics::calculateHeight()` function is called to obtain height information (described in detail in the next sections). Consequently, received controller input is evaluated in the manner described in section 3.4 on gradual leaning, the result of which gives us the current steer angle δ. After that, fatigue and maximum power are calculated as outlined in section 4.2 about the physiological properties of the rider. With known maximum power, steer angle and height information, acceleration and speed can be evaluated in the manner described in section 2.6. Finally, the new position and direction are set, including yaw and lean.

The engine uses a wide range of variables. Several of them come in pairs, as it is often necessary to know their values in the previous timestep – most importantly time, speed, position and direction. Only position and direction are represented as vectors (except for the colours for the fatigue and lean indicators which are held in RGB). In these vectors, x and z denote the movement on a flat plane and the y component holds the value for height, as is commonly the case in 3D modelling software. It was sufficient and more practical to hold all other values as floating point numbers.

Evaluation closely follows the equations presented in chapter 2 with the exception of position, which requires slight clarification as to how actual world-space coordinates are obtained. In section 2.3.1 we calculate the values of x and y that are illustrated in figure 2.3, but these are parallel to the direction and yet have to be transformed into world x and z coordinates to modify the x and z components of the position vector which is also in world space coordinates. The equations used follow.

$$p_x = p_x + \operatorname{sgn}(\Delta\psi)\sin(\arctan\frac{x}{y} - \psi)\sqrt{x^2+y^2}$$
$$p_z = p_z - \operatorname{sgn}(\Delta\psi)\cos(\arctan\frac{x}{y} - \psi)\sqrt{x^2+y^2}.$$

6.2 Collisions

Height information for collisions is retrieved from a heightmap that gets generated from the track 3D model at application startup. Originally, I wanted to generate the heightmaps

beforehand and only load them into the simulator. When rendering in Maya to the Maya IFF format, it is possible to set the renderer to include Z-buffer data in the output file. Although Z-buffer data is recorded as 32 bit floats, there is no easy way of extracting the data. The IFF format is binary and practically only fully supported by FCheck, a program that comes bundled with Maya. FCheck allows to view both the rendered image and the Z-buffer data, but is only able to export a grayscale image[1]) with 8 bit precision, which is not at all sufficient.

Fortunately, it is possible to render the track in OpenGL with an orthographic view and use the `glReadPixels()` function to retrieve Z-buffer information with high precision. The only limitation this poses is the actual screen resolution, as OpenGL will not render off-screen, even if window size exceeds the size of the screen. I bypassed this limitation by making 4 separate renders at a lower resolution (512x512) and stitching them to create a 1024x1024 heightmap which has proved sufficient for the task. The resolution could of course be increased or heightmaps could even be generated dynamically based on the rider's position to save memory.

The heightmaps do have their weak spots – I observed that they did not perform well with surfaces that were not perfectly smooth. I first observed this when testing the velodrome track, but smoothing the original curves and regenerating the surface greatly improved performance. On the other hand, increasing heightmap resolution made no improvement. I must admit that minor irregularities are still present in certain parts of the banking, but I adjudicate this more to the 3D model than to the generated heightmaps themselves. Also, most roads are not perfectly smooth, not even a concrete velodrome track is and a rider can feel all the bumps very well. Therefore it can even be viewed as a reality-enhancing feature.

6.2.1 Height Information Extraction

Height information is required for determining the rider's position on the y-axis, the bicycle's pitch and consequently slope resistance (section 2.5.3), and finally for checking whether the rider has ridden into an obstacle or not. The `Physics::getHeightAt()` function is responsible for acquiring the height values – it takes two floating point parameters, the x and z position coordinate in world space and uses bilinear interpolation to approximate the result based on the four surrounding discrete positions on the heightmap.

In every cycle, three height values are retrieved automatically: at rear and front wheel ground contact for position and pitch, and at the furthermost tip of the front wheel to determine a collision.

6.2.2 Collision Detection

Collision evaluation is initiated if the height at the tip of the front wheel is elevated by more than 0.2m opposed to the front wheel ground contact patch, or if the tip is more than 0.5m below. If the tip is more than 0.2m above, 3 situations can occur.

1. If the angle between the bicycle and obstacle is lower than 15°, the rider is only slowed down and shifted away from the obstacle. Though this is very unrealistic, I believe it adds a reasonable degree of comfort for the user. The angle is determined by testing heights 15cm in front of the tip of the wheel at 15° on either side.
2. If the speed is lower than $2\text{m}\cdot\text{s}^{-1}$, the rider is "bounced back". This is relatively realistic – if a rider rides into a wall at a small speed, material flexibility makes him bounce off it.

[1]) In fact, and standard RGB image is exported, but it is created by converting a 8 bit greyscale image, mainly to ensure compatibility.

3. In all other cases the result is inevitably a fall over the handlebars. After such a fall the rider's position is reset to the initial position on the map.

If the tip is lower than 0.5m below front wheel ground contact, situation 3 above automatically takes place.

In situations 2 and 3, simulation stops and evaluation is handed over to the `Physics::fallAnim()` function instead. For each case, this function has a predefined sequence of transformations to perform within three seconds. After that, standard simulation restarts awaiting user input.

6.2.3 Sideslip Evaluation

Sideslip also requires height information because it depends on the banking angle of the track, as illustrated in section 2.4.1. Sideslip is evaluated automatically after collisions are negative, using height values 5cm on either side of the rear wheel's position, perpendicular to the bicycle's forward axis. The equations for evaluation were presented in section 2.4.1. Once again, if sideslip occurs and the rider is to fall, `Physics::fallAnim()` computes the fall animation and the rider's position is reset after three seconds.

Chapter 7
Rendering Framework

Code that can be considered as the rendering framework spans through the whole project. Basic functions such as OpenGL context creation or window management are defined in `main.cpp`; `Shader.cpp` holds the `Shader` class responsible for shader loading and compilation, and lastly the `FbxMeshNode` class in `FbxHandler.cpp` contains routines for vertex data manipulation, their passing to the shader programs and rendering itself.

I chose to target the application at OpenGL 3.1 so that it could run on older hardware, because I did not intend to use any of the novel features offered solely by newer versions of OpenGL. Nonetheless, no deprecated functions have been used and the application is fully forward compatible with OpenGL 4.3. It has been tested on a desktop system with an Intel Core 2 Duo E6300 processor, 6GB of RAM and a Nvidia GeForce GTS 450 graphics card and a Thinkpad X220 laptop with an Intel i5-2520M processor, 4GB of RAM and an integrated Intel HD Graphics 3000 graphics adapter with no major performance issues on either system. The polygon count of the tracks is probably too high and accounts for most GPU processing power used. Ideally, this could be fixed by using several levels of detail according to distance from the cyclist. Texture downsizing greatly improved both GPU processing power and GPU memory usage.

7.1 Libraries

First, the GLFW library[1]) provides us with functions to create and manage an OpenGL context along with opening and manipulating with a window. The library also facilitates keyboard and controller input. Callback functions are defined to handle window resizing and keyboard input, and the `Controller` class manages controller input – the `Physics` class uses an instance of it to poll controller position when needed. The GLFW Reference Manual [13] provides descriptions of all functions GLFW offers. The GLEW[2]) library is used in conjunction with GLFW to load OpenGL extensions.

GLM[3]) contains convenient mathematical functions aimed to suit both OpenGL and GLSL conventions and so this library is used laboriously throughout the whole application for matrix and vector operations, goniometric functions and much more.

7.2 Vertex Data Loading

Because vertex data manipulation is closely connected with the actual model loading, it is done within the `FbxHandler` and `FbxMeshNode` classes. The `FbxHandler`'s constructor generates one vertex array object to be further used within the application. For every FBX scene (file), an instance of the `Buffers` class is created – this creates vertex buffer objects and variables used for their indexing. Two vertex buffer objects are used per scene – one

[1]) `http://www.glfw.org/`
[2]) `http://glew.sourceforge.net/`
[3]) `http://glm.g-truc.net/`

for vertex data (vertex positions, normals and UVs) and the other for indices. Functions `FbxMeshNode::doSomethingWithMeshNew()` and `FbxMeshNode::prepare_deform()` are responsible for extracting vertex and deformation data from the FBX scene graph and storing them in user memory in a suitable manner for the later use of the OpenGL `glBufferSubData()` function. This has to be done in advance to determine the total size of the data to set the correct size of the buffer via `glBufferData()`, prior to filling the buffer with seperate `glBufferSubData()` calls in each `FbxMeshNode`. All is done recursively, starting with the scene's root node. Recursion is always handled in the appropriate prototype functions of the `FbxSceneNode` class.

Deformation requires two transform matrices per bone (cluster) – one that does not change and defines the mesh's position in respect of the bone and the other that can change in every timestep and represents the bone's transformation due to animation at a given time.

During this process, transformation matrices are also created in the `FbxSceneNode` constructor. Each `FbxSceneNode` and each derived `FbxMeshNode` uses four transformation matrices:

- `Model` is retrieved from the FBX scene and represents the node's position within the scene
- `fixed_world` is set at scene loading and is mainly used to scale the scenes to proportionally fit, it is identical throughout each scene and usually does not change
- `temp_world` can change in every simulation timestep if necessary – it is predominantly used to set the bicycle's position. Once again, it is identical for all nodes in a scene
- `temp_local` can also change frequently and is unique for each node, its use is mainly for animations such as wheel rotation.

All of the matrices are of `glm::mat4` type. The resulting model matrix is calculated in the `FbxMeshNode::Render` function by matrix multiplication.

```
temp_world * fixed_world * Model * temp_local
```

If the material associated with the mesh contains a texture, it too is loaded in this phase. The GLFW library provides functionality to directly load textures in the TGA image format, which I make use of.

7.3 Rendering

Similar to vertex data loading, rendering is also done in a recursive manner. After rendering is called in an instance of the `FbxHandler` class, `FbxHandler::render()` walks through the list of root nodes that are associated with the instance and calls their `FbxSceneNode::Render()` functions. These recursively traverse each scene and `FbxMeshNode::Render()` handles the rendering of each mesh. Based on which features each `FbxMeshNode` incorporates (a texture and/or a deformer), appropriate vertex attribute arrays are enabled and their layout set by the the `glVertexAttribPointer` function.

Apart from vertex data and a model transform matrix, discussed in the previous section, shaders also require a view and projection matrix for correct lighting and positioning in camera space. The GLM library comes to use here once again and provides a convenient `glm::lookAt()` function for the view matrix and `glm::perspective()` for the projection matrix. These are sent as parameters to the rendering functions.

Lastly, section 6.2 discussed heightmaps and indicated the use of OpenGL for their generation. These we obtain by setting the projection matrix to orthographic view by using the

`glm::ortho()` function and rendering the scene containing the track. The render contains Z-buffer data which can be extracted by the `glReadPixels()` function with the use of `GL_DEPTH_COMPONENT` as the format parameter.

7.4 GLSL Shaders

As loading and compiling shaders is a routine task, I decided not to reinvent the wheel and used code that is presented at `http://www.opengl-tutorial.org/`. It is offered for unlimited use with a WTFPL Public Licence[1]). The code is converted into the `Shader` class defined in `Shader.cpp` with the addition of necessary variables, such as IDs of uniform variables used in the shaders. An instance of this class can then be created from throughout my program with different shaders if necessary. The simulator uses two sets of shaders, one for ordinary scene rendering and the other for the heads-up display.

7.4.1 General Rendering

The general scene rendering shaders are based on those I used in my semestral project in the Computer Graphics course (A7B39PGR). Although their outline was provided in course materials, they still required a student's own input to render correctly.

For the thesis, I expanded the features offered by the vertex shader to support mesh deformation. Vertex-to-bone transform matrices as well as bone animation matrices are uploaded to the shader as arrays of uniforms, supporting up to 100 bones. Apart from position and normal vectors of each vertex, weights and indices to these matrices are also sent as a `vec4`, hence the 4 bones per vertex limitation. The model transformation matrix for each vertex is then calculated as

$$B = \sum_{j=0}^{3} w_j A_{i(j)} T_{i(j)}$$

where w_j denotes weights and $i(j)$ indices to the arrays of bone animation and mesh to bone transformation matrices A and T respectively.

The fragment shader allows to create directional and point lights and uses Gouraud shading to compute the resulting fragment colour.

[14] helped with reviving my knowledge on this topic and provided a better understanding of shader programming principles.

7.4.2 Heads-up Display

The HUD incorporates very simple vertex and fragment shaders, `textv.vs` and `textf.fs`. The purpose of the vertex shader is to calculate the positioning of the indicators in the lower right corner of the screen; the fragment shader textures the drawn shape (4 triangles that make up a rectangle) with a predefined image, using its alpha channel to visualise the two letters. A uniform colour vector is set to determine the colour of each of the letters.

[1]) Licence available at `http://www.opengl-tutorial.org/download/`

Chapter 8
Video Comparison

The thesis includes five video comparison clips that all consist of two video frames next to each other, the left showing real life footage and the right a screen capture from the simulator. The aim was to depict similar situations the user can achieve in the simulator so I chose to record the footage at the Třebešín velodrome in Prague. Compared to the velodrome in Brno (the track in the simulator copies its geometry) the track in Prague is shorter with only 333.33m in circumference opposed to 400m. Therefore it was not possible to exactly synchronise the clips as I did not want to alter the playback speed to maintain authenticity, but I did my best to find matching clips. I recorded the real life footage to match the 1st and 3rd person views available in the simulator and added two more, mainly to show a cyclist from a non-riding perspective. I did not include the other views in the simulator because they were highly impractical for normal use. The video files can be found in the `/video/` directory of the CD:

- `back.mp4` resembles the 3rd person view and was recorded from a bicycle riding behind me
- `bars.mp4` shows the situation in 1st person view, the camera was attached to my body at a similar position as in the simulator
- `front.mp4` was recorded from a bicycle riding in front – though straightforward to film, quite challenging to perform in the simulator without seeing the track ahead
- `lean.mp4` demonstrates turning and leaning, shown in 3rd person view
- `static.mp4` shows one lap of the track from a static position and a detail of the bend from above.

Figure 8.1. A video snapshot taken from `front.mp4`

A simulator attempts to reflect reality, but reality is full of an indefinite number of details and that is why it is impossible to do so perfectly. It is apparent from the videos that graphically the simulator lacks some of these details such as complex lighting or shadows and of course a more sophisticated surrounding environment. Though as far as physics are concerned, I believe the videos prove that the attempt on creating a bicycle simulator was successful.

Chapter 9
Discussion

Like I suggested in the Introduction (1.2) there are not many implementations my project could be directly compared with. The closest is probably Mountain Bike Adrenaline, but that is still a full feature computer game produced by a game studio and a team of people. My Bicycle Simulator only provides a starting point yet to be improved to match current state of the art production.

I consider the most limiting factor to be the graphics – I must admit that the results in this section cannot be considered perfect, but modelling and animation is a very time consuming occupation and the time required to produce a detailed and good looking result is largely disproportional to that required for a model providing basic functionality. I estimate that creating the tracks at a level found in modern racing games would easily supersede the effort needed for all other parts of the thesis. On the other hand, game studios employ teams of both 3D and 2D graphics professionals with whom I cannot compare myself neither in number nor skills. The framework though does not pose limits to what can be rendered, so any superior modelled track can be easily added to the simulator in FBX format. The same goes for the rider and bicycle if it follows the same structure as the original. A graphically enhanced HUD – possibly in the design of a cyclometer – together with more indicators would also provide the user with more comfort.

If also extended by a realistic control device as proposed in section 3.5, the simulator could serve as a training tool similar to Tacx (section 1.2). It could even be modified to be used for visualisation during Goldsprints[1]), possibly using existing hardware already in use for performance monitoring during the races.

The physics engine itself so far proves sufficient even though it does not implement some advanced features I initially thought I would cover. Namely, these include individual brake control, which would facilitate the option of skidding the rear wheel and introduce the danger of falling across the bars when applying too much braking power to the front wheel. Another feature was forward and backward lean, but it is discussable whether it is at all possible to simulate it in such a way for it to actually be useful. Mountain Bike Adrenaline for example includes these controls but I never saw any actual difference in handling when used. In section 6.2 about collisions I mentioned the somewhat suboptimal performance of heightmaps in certain cases when their quality was largely affected by the evenness and smoothness of the original 3D models which they were created from. This is a limitation that can be either addressed by only creating tracks that are sufficiently smooth or by attempting to develop a superior height information retrieval system.

A feature that I haven't yet mentioned at all is sound. It is unquestionable that sound would enhance the visual experience – chain rattling, wind and the rider's breathing rhythm – but it was not the main focus of this thesis.

[1]) Goldsprints are a form of head to head racing on bicycle rollers, originating in the bicycle messenger subculture. They usually take place at social events as supporting programme.

Chapter 10
Conclusion

In the thesis, I created a simulator that enables its user to control a rider on a bicycle on three different tracks. Completion of the project required good understanding of a wide range of fields from physics through 3D modelling to programming itself.

Firstly, I presented a theoretical insight into how real world physics affect a bicycle ride and provided equations that were further necessary for developing the physics engine. I experimented with two different controllers and outlined a design for a controller based on a real bicycle that could be constructed in the future. I concluded that controller precision and sensitivity is the most limiting factor for human interaction with the simulator.

A large part of my work on the thesis consisted of 3D modelling and animation which significantly helped me to improve my skills in the field, as for being my current primary interest in my studies. For manipulation with the models, I implemented a 3D model loader for the Autodesk FBX format. Apart from geometry data, it supports skinning and skeletal animation which were required for the project.

I used the conclusions from the section on bicycle physics to develop a physics engine for the simulator and assembled the aforementioned modules into a C++ application with the use of OpenGL to render on screen and provide the user with the experience of the Bicycle Simulator.

Finally, I recorded a series of videos to make a comparison between cycling in real life and within the simulator. It is upon the user to decide on how well the simulator performs, but based on the video comparison I believe it meets the expectations.

References

[1] J. K. Moore. *Human Control of a Bicycle*. UC Davis Doctoral Dissertation, 2012. http://moorepants.github.com/dissertation/.

[2] J. P. Meijaard, Jim M. Papadopoulos, Andy Ruina, and A. L. Schwab. Linearized dynamics equations for the balance and steer of a bicycle: A benchmark and review. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, August 2007.

[3] David Gordon Wilson with contributions by Jim Papadopoulos. *Bicycling Science*. The MIT Press, third edition, 2004.

[4] Ing. Karel Cibula. *Mechanika jízdního kola*. Vydavatelství ČVUT, 1996.

[5] J. D. G. Kooijman, J. P. Meijaard, Jim M. Papadopoulos, Andy Ruina, and A. L. Schwab. A bicycle can be self-stable without gyroscopic or caster effects. *Science, Volume 332, Issue 6027, pp. 339-*, April 2011.

[6] Jr. Parker, J. F. and V. R. West. *Bioastronautics Data Book*. NASA SP-3006, second edition, 1973.

[7] Ramondo Spinnetti. Backwards versus forward pedaling: Comparison tests. *Human power 6, no. 3:1, 10-12*, 1987.

[8] Autodesk Inc. *Autodesk Maya 8.5 Character Setup*, 2007.

[9] Lawrie Douglas. Olympics watch - the velodrome. *Engineering and Technology Magazine, vol 5, issue 2*, February 2010. http://eandt.theiet.org/magazine/2010/02/olympics-watch.cfm.

[10] Chris Nadovich. A new generation. http://www.nadovich.com/chris/track.

[11] Patrick Cozzi and Cristophe Riccio. *OpenGL Insights*. CRC Press, 2012.

[12] Autodesk Inc. *Autodesk FBX 2013 SDK Documentation*, 2012. http://docs.autodesk.com/FBX/2013/ENU/FBX-SDK-Documentation/index.html.

[13] Marcus Geelnard and Camilla Berglund. *GLFW Reference Manual*, March 2013. http://www.glfw.org/GLFWReference278.pdf.

[14] Mike Bailey and Steve Cunningham. *Graphics Shaders*. CRC Press, second edition, 2012.

Appendix A
Implementation

The simulator was developed using Microsoft Visual Studio 2010. The source code as well as a build for Microsoft Windows are in the `/simulator/` directory on the enclosed CD.

A.1 Files and Classes

- `FbxHandler.cpp` and `FbxHandler.h`
  - `class Buffers`, responsible for creating and manipulating with vertex buffer objects
  - `class FbxHandler`, handles the loading and management of FBX scenes
  - `class FbxMeshNode:FbxSceneNode`, derived from `FbxSceneNode`, provides mesh-specific functionality to the scene graph
  - `class FbxSceneNode`, basic scene graph node
- `main.cpp` and `main.h` holds the `main()` function with the core application cycle and other initialisation functions
  - `class Controller`, provides access to the controller through the GLFW library
  - `class HUD_display`, responsible for rendering the heads-up display
  - `class Physics`, encapsulates the physics engine
- `Shader.cpp` and `Shader.h`
  - `class Shader`, for shader loading and compilation

A.2 Compilation

The following libraries are used and are required for compilation. Except for the Autodesk FBX SDK, all are included on the CD. The FBX SDK can be downloaded at `http://usa.autodesk.com/adsk/servlet/pc/item?siteID=123112&id=10775847`

- GLFW version 2.7
- GLEW version 1.9.0
- GLM version 0.9.4.1
- Autodesk FBX SDK 2013.3

A.3 User Manual

Once started, the system is checked for a controller. If none is present, the user is prompted with a message and is asked to attach one and restart the application. Otherwise, the command prompt window briefs the user to choose the controller mode.

- Option S starts the simulator in steering wheel mode, setting handlebar threshold at 45° on either side.

- Option J starts the simulator in joystick 4-axis mode with a major and minor turning axis and a sensitivity slider, as discussed in section 3.1.
- Option K starts in joystick 2-axis mode with handlebar threshold at 10° on either side.

Next, the user is prompted to pick the track.

- Option V loads the velodrome.
- Option B loads the road with bends.
- Option O loads offroad terrain.

After that, the simulator loads and the bicycle ride can start. Details about controllers can be found in chapter 3.

Additionally, pressing 'C' on the keyboard changes the camera from 1st person to 3rd person view and back.

Pressing '1', '2' or '3' on the keyboard selects gears as outlined in section 4.2.

There are two indicators in the lower right corner of the screen: the letter 'F' symbolising fatigue and 'L' lean. Their colour indicates their state, green being OK and red meaning exhaustion for fatigue and critical danger for lean.

In the command prompt window the user can monitor speed, maximum available power and an exact reading of fatigue.

Appendix B
Enclosed CD

The CD as part of the thesis is enclosed with the printed written part. It has the following directory structure:

- `/matlab/` contains the Matlab script for velodrome curve generation
- `/models/` contains the original Maya models that were later exported into FBX. The folder follows a standard Maya project structure
  - `scenes/` contains the 3D models themselves
  - `sourceimages/` contains original, full-resolution textures
- `/simulator/`
  - `bin/` contains the compiled version of the simulator along with necessary DLLs and shaders
    - `images/` contains images required for the simulator to run
    - `models/` contains the 3D models in FBX format
      - `sourceimages/` contains downsized texture files for the FBX 3D models
  - `src/` contains the source code as a Microsoft Visual Studio 2010 project
- `/thesis/`
  - `pdf/` contains the final version of the written part of the thesis in PDF format.
  - `tex/` contains the TEX source files and images for the thesis. The document is written in plain TEX and uses the CTUstyle template by Petr Olšák, available at `http://petr.olsak.net/ctustyle-e.html`
- `/video/` contains video samples with comparison footage recorded in real life and in the simulator

Appendix C
List of Symbols and Abbreviations

C.1 Symbols

Symbol	*Meaning*
w	wheelbase
c	trail
λ	head tube angle
ψ	direction, or in flight terminology "yaw"
ϕ	lean, or in flight terminology "roll"
θ_B	slope angle, or in flight terminology "pitch"
δ	steer angle
θ_R	rear wheel rotation
θ_F	front wheel rotation
θ_C	crank rotation
ζ	turning angle
r	radius
ω	angular speed
v	speed
t	time
F_c	centripetal force
F_g	gravitational force
g	gravitational acceleration on Earth $\approx 9.81\text{m}\cdot\text{s}^{-2}$
μ	coefficient of friction
C_B	drivetrain and bearing loss coefficient
C_{rr}	rolling resistance coefficient
R_r	rolling resistance
R_s	slope resistance
R_D	air drag
ρ	environment density
C_D	drag coefficient
S	cross-sectional area
p	pressure
R_{specific}	specific gas constant
T	thermodynamic temperature
P	power
a	acceleration
F_P	propulsion force
f	fatigue
f	frequency (pedalling cadence)
α	angular acceleration

C.2 Abbreviations

Abbreviation	*Meaning*
API	application programming interface
COLLADA	a 3D graphics interchange format, derived from "collaborative design activity"
DLL	Dynamic-link library
EPS	Encapsulated PostScript, a vector graphics format
FBX	a 3D graphics interchange format, derived from the name of its original native application "Filmbox"
FK	forward kinematics, an animation technique
GLEW	OpenGL Extension Wrangler Library
GLFW	a library for creating and managing a OpenGL contexts
GLM	OpenGL Mathematics, a library providing mathematical functions frequently used in computer graphics
GLSL	OpenGL Shading Language
HUD	heads-up display
IFF	Maya Image File Format, the default image format for renders in Maya
IK	inverse kinematics, an animation technique
iOS	a mobile operating system developed and distributed by Apple Inc.
MD5	a 3D graphics interchange format
NURBS	non-uniform rational basis spline, a mathematical model for creating curves and surfaces in computer graphics
OBJ	a 3D graphics interchange format
OpenGL	Open Graphics Library, an API for rendering 2D and 3D computer graphics
RGB	red, green, blue – the standard sequence of basic colours
SDK	software development kit
TGA	Truevision TGA or TARGA, a raster image file format
UV	as in UV mapping, a way of projecting a 2D image onto a 3D object. In the process u and v coordinates are commonly used instead of x and y in order to prevent confusion